SHARP

SHARP
COMPUTER
SOFTWARE

X68000用

C COMPILER ver2.0 PRO-68K

Cライブラリマニュアル VOL.1

X68000用

C COMPILER PRO-68K ver2.0

Cライブラリマニュアル VOL.1

SHARP

はじめに

本書は、「C compiler PRO-68K ver2.0」(以下、XC コンパイラと表記します)でプログラム開発を行うときに必要となるCライブラリの概要と使用方法、小文字関数リファレンスに関して記述しています。
大文字関数のリファレンスに関しては、「Cライブラリマニュアル VOL.2」をご覧ください。

なお、別冊の「Cユーザーズマニュアル」に XC コンパイラの商品構成、動作環境の作成方法、各マニュアルの内容が説明されています。
初めて本プログラムをご使用になる方は、必ず「Cユーザーズマニュアル」を先にご覧ください。

本書は、C言語について一定以上の知識を有し、プログラミング経験のあるユーザーが、Cライブラリの機能と使用方法を理解して X68000 の豊富な機能をフルに活用できるようになること、さらに進んでユーザーが自分自身でもライブラリを作成できるようになることを目的として書かれています。

なお、C言語の書式・文法については「Cリファレンスマニュアル」、開発環境の設定や開発手順などについては「Cユーザーズマニュアル」を参照してください。

Cライブラリの概要

Human68k のCランタイムライブラリは、C言語でプログラム開発を行う際に必要となる関数およびマクロで構成されています。
C言語では、データの入出力などハードウェアに依存する処理は、言語仕様からはずし、これを関数として提供しています。
また、文字列操作、およびサーチやソートといった、使用頻度の高い汎用的な処理についても関数として提供し、Cプログラマの労力を軽減しています。
Human68k のCランタイムライブラリでは、上記のようなC標準ライブラリの他に、日本語処理をサポートする関数や、IOCS コールライブラリ、DOS コールライブラリ、BASIC ライブラリといった、Human68k 固有の関数が豊富に用意され、その処理能力はきわめて強力なものとなっています。

たとえば、IOCS コールライブラリによって、キーボードやプリンタといった標準的な入出力機器だけでなく、ジョイスティック、RS-232C、ADPCM、FM 音源、マウス、グラフィック、スプライト機能といった、多様な入出力操作が可能となります。
また、DOS コールライブラリによって、日本語処理、日付・時刻の制御、環境変数の操作などの、Human68k オペレーティングシステムの機能を有効に利用することができます。
BASIC ライブラリは、BASTOC において使用し、BASIC 言語からC言語への変換を容易にしています。
また、これらの関数をサポートするために、専用のインクルードファイルが用意されています。

Cランタイムライブラリの項目別の分類、およびその項目に属する関数の名前と機能については、「第3章　項目別ランタイムルーチン」を参照してください。

本書の構成

本書は以下の内容から構成されています。

第１章　ライブラリルーチンの使いかた

ライブラリルーチンの基本的な使い方、および関数とマクロの違い、ライブラリとともに使用するインクルードファイル、エラー処理など、各ライブラリに共通する項目について述べます。

第２章　インクルードファイル

ライブラリ使用時に必要となるインクルードファイルの概要、およびその使用法について述べます。
また、Ｃライブラリで使用する型とマクロ、グローバル変数および、標準型の構造体についても述べます。

第３章　項目別ランタイムルーチン

Ｃライブラリを項目別に分けて、ライブラリの種類とその機能について説明します。

第４章　リファレンス（小文字）

各ライブラリのリファレンスであり、実際にライブラリを使用するときに使用手引として利用してください。

付録

ユーザーライブラリの作成方法を、具体的な例にそって説明します。

CONTENTS

第1章　ライブラリルーチンの使いかた

第2章　インクルードファイル

第4章 リファレンス

付 録 ユーザーライブラリの作り方

索 引

関数索引

機能別関数索引

第 1 章

ライブラリルーチンの使い方

関数とマクロ
インクルードファイル
エラーハンドリング
ライブラリファイル

本章では、Ｃランタイムライブラリを使用するにあたって注意しなければならないこと、ライブラリの使用環境、ライブラリの設定方法などについて説明します。

1.1 関数とマクロ

Cランタイムライブラリは、C言語、あるいはアセンブラで作成された関数であり、一般の関数と同じようにプログラムの中から呼び出すことができます。
同時に、これらのライブラリのうちのいくつかは、マクロとしても用意されています。
マクロとは、プリプロセッサ命令#defineで定義する、値や式を表す識別子のことです。
関数もマクロも引数をまったくとらないか、または、マクロ定義で仮引数に置き換え可能な引数を定義することができます。
いいかえれば、マクロも関数と同じように、引数をとり、値を返すことができます。
関数とマクロの違いは、実行速度とメモリサイズという点にあります。
たとえば、マクロはコンパイル前にプリプロセッサによって展開されるので、関数呼び出しの際にみられるようなオーバーヘッドはありません。
しかし、マクロは一度しか定義されない関数とは異なり、呼び出されるたびに展開されるので、必然的にメモリサイズが大きくなります。
このため、Cランタイムライブラリでは特性に応じて選択できるように、関数定義とマクロ定義の双方をもつライブラリルーチンがいくつかあります。
なお、マクロの中には、toupperルーチンのように、引数を2回以上評価する場合、副作用が発生して結果が正しく返らないものもありますので注意してください（toupperルーチンの使用上の注意については、「3.3文字の分類と変換」およびtoupperルーチンのリファレンスを参照してください。
また、使用にあたって注意が必要となるものについては、そのつど説明します）。

1.2 インクルードファイル

Cランタイムライブラリの関数、またはマクロを呼び出すにあたっては、それらの関数やマクロを外部宣言し、(関数やマクロで使用する)構造体や記号定数を定義しているインクルードファイルが必要となります。
必要となるインクルードファイルは、「第3章　項目別ランタイムルーチン」または、「第4章　リファレンス」に記載していますので、これに従い、#includeプリプロセッサ命令を使ってソースファイルにとり込んでください。

1.3 エラーハンドリング

Cランタイムライブラリの関数のほとんどは戻り値を返しますが、この戻り値によって、処理が正常に終了したか、あるいは異常終了したかが判定できるようになっています。
関数を呼び出したときは必ず戻り値の判定を行うようにしてください。
関数が異常終了したにもかかわらず、戻り値の判定を行わずに処理を続行した場合、プログラムが暴走することもあります。

Cランタイムライブラリの関数の中には、エラーが発生した場合にグローバル変数errnoにエラーの詳細コードを設定してくれるものもあります。
このような関数を使用するにあたっては、インクルードファイルerror.hをソースファイルにとり込むことにより、errnoを参照し、エラーコードのチェックを行うことができます。

また、Cランタイムライブラリの中には、エラー処理を行う関数がありますので、これを以下に示します。

perror関数 ………………エラーメッセージを標準エラー出力(stderr)に表示します。

matherr関数 ……………数値演算で発生したエラーの処理を行うため数値演算関数自身が呼び出します。
また、ユーザー自身でmatherr関数を作成し、使用することができます。
その場合には、matherr関数のリファレンスに明記してある規約に従ってください。

ferror関数 ………………ストリーム入出力におけるエラーをチェックします。

clearerr関数………………ストリームのエラーインジケータおよびエンドオブファイルインジケータをクリアします。

1.4 ライブラリファイル

ライブラリファイルには、オブジェクトファイル、ライブラリファイル、およびアーカイブファイルという3つのファイル形式があります。

オブジェクトファイルは、Cコンパイラによって出力される1個のオブジェクトコードを格納しておくファイルであり、拡張子はoです。

これに対してライブラリファイル、およびアーカイブファイルとは、複数個のオブジェクトファイル、またはソースファイルを1個のファイルにまとめて登録したものであり、拡張子はlまたはaとなります（ただし、1個のアーカイブファイルに、オブジェクトファイルとソースファイルを混在させて登録することはできません。ライブラリファイルやアーカイブファイルの詳細については、「アセンブラマニュアル」の「第1部　第8章　ライブラリアン」と「第1部　第7章　アーカイバ」をそれぞれ参照してください。なお、オブジェクトファイルは、ライブラリアン（LIB）を使用してライブラリファイル（拡張子 l）へ、ソースファイルはアーカイバ（AR）を使用してアーカイブファイル（拡張子 a）へ登録します。
なお、XCの標準ライブラリの登録ファイル名はCLIB.lです。

第2章

インクルードファイル

インクルードファイルについて
型とマクロの定義
グローバル変数
標準型
インクルードファイルの種類

本章では、インクルードファイルについて説明します。
フォーマットの定まった変数や構造体が、あらかじめ定義されているファイルをインクルードファイルといいます。
XCで特定されている、このインクルードファイルにおける、グローバル変数と呼ばれる変数、標準型と呼ばれる構造体名、ならびにインクルードファイルの種類についても解説します。

2.1 インクルードファイルについて

Cのプログラムを作成するにあたって、ユーザーは必要な処理手順、変数等をコーディングしなければなりませんが、たとえば、プログラム間で共通に使用される（フォーマットの定まった）構造体や、変数をユーザープログラムでいちいち定義しなければならないとしたら、プログラム開発の生産性が低下するだけでなく、コーディングミスの多発、といった事態も避けられません。

このようなわずらわしい作業から、プログラマを解放するためにXCでは、上記のような構造体や、変数をあらかじめ特定のファイル中に定義してあります。

このようなファイルのことを、インクルードファイルといいます。
なお、インクルードファイルの拡張子は習慣的にｈにします。
ユーザーは、特定の構造体や変数が必要になった時点でこれをコーディングする代わりに、該当の構造体や変数が定義されているインクルードファイルを、ソースプログラムから呼び出します。
また、ライブラリの中には、これを呼び出すにあたって特定のインクルードファイルが必要となるものもあります。

Cランタイムライブラリ関数の宣言には引数型リストを含むものと、含まないものの2種類があります。

関数の宣言は、プリプロセッサの#ifdef、または#ifndefに含まれています。
XCはデフォルトでは、引数型リストを含む関数宣言が生成、コンパイルされ、引数型チェックが行われます。
（__STDC__識別子が1に定義されています）

2.1 インクルードファイルについて

引数型のチェックを行わないようにするには、次の方法があります。

1. CC コマンドラインのXスイッチを使用して、__STDC__識別子を未定義にします。
2. ソースプログラム中で、#undef プリプロセッサ命令を使用して__STDC__の定義を取り消します。
   この場合、#undef は# include の前になければなりません。

インクルードファイルの呼び出しは、#include プリプロセッサ命令により行います。
呼び出し手順を以下に示します。

① #include 〈ファイル名〉
② #include "ファイル名"

①の場合は以下の順で検索します。

環境変数 include が指すディレクトリ
↓
カレントドライブのカレントディレクトリ下の include サブディレクトリ
↓
カレントドライブのカレントディレクトリ

②の場合は

カレントドライブのカレントディレクトリ
↓
環境変数 include が指すディレクトリ
↓
カレントドライブのカレントディレクトリ下の include サブディレクトリ

また、ファイル名は大文字・小文字を区別しません。
なお、ユーザー用のインクルードファイルをユーザー自身で作成することもできます（作成はエディタによって行います）。

2.1　インクルードファイルについて

インクルードファイルの呼び出し例を以下に示します。

例：
```
#include <stdio.h>
#include <ctype.h>
#include "mylib.h"

main(int argc, char *argv[])
 {
         :
         :
 }
```

2.2 型とマクロの定義

インクルードファイルには、頻繁に使用される値や、動作環境や機種に依存するような型を定義してあります。

ここで定義してある型やマクロを使用すると、ソースファイルを他のCコンパイラに移植した場合でも、変更は必要最小限で済むようになります。

定義されている型の種類とその内容は以下の通りです。

型	ファイル	内　容
div_t	stddef. h	intの剰余を求めるdiv関数の返り値を格納できる型
ldiv_t	stddef. h	longの剰余を求めるldiv関数の返り値を格納できる型
ptrdiff_t	stddef. h	ポインタ同士の引き算の結果が正しく格納できる型
size_t	stddef. h	バイト数やsizeofの返す値が正しく格納できる型
time_t	stddef. h	時間やtime関数などの返す値が正しく格納できる型
clock_t	stddef. h	秒数やclock関数などの返す値が正しく格納できる型
fpos_t	stddef. h	fgetpos関数などに与えるファイルの位置を表す値を正しく格納できる型
INT	class. h	int型
UINT	class. h	unsigned int型
WORD	class. h	short型
UWORD	class. h	unsigned short型
BYTE	class. h	char型
UBYTE	class. h	unsigned char型
VOID	class. h	void型
DEFAULT	class. h	int型

2.2　型とマクロの定義

型	ファイル	内　　容
BOOLEAN	class. h	int型
LONG	class. h	long型
ULONG	class. h	unsigned long型
jmp_buf	setjmp. h	setjmp関数やlongjmp関数で使用されるプログラム環境を格納できる型
va_list	stdarg. h	va_startマクロ関数やva_argマクロ関数で使用する、可変引数を指し示す変数の型
FILE	stdio. h	ストリーム入出力用の構造体を表す型

定義されているマクロの種類とその内容は以下の通りです。

マクロ	ファイル	内　　容
CLK_TCK	stddef. h	clock関数が返す値から秒にする場合に割り算する値
CLOCKS_PAR_SEC	stddef. h	clock関数が返す値から秒にする場合に割り算する値
SEEK_SET	stddef. h	fseek関数で、先頭からシークする場合のモード
SEEK_CUR	stddef. h	fseek関数で、現在位置からシークする場合のモード
SEEK_END	stddef. h	fseek関数で、最後からシークする場合のモード
NULL	stddef. h	ヌルポインタを表す値
EOF	stddef. h	ストリーム入出力関数がエラーの時返す値
EOS	stddef. h	文字列の終わりを表す文字
EXIT_SUCCESS	stdlib. h	プログラムが正常終了する場合にexit関数に指定する値
EXIT_FAILURE	stdlib. h	プログラムが異常終了する場合にexit関数に指定する値
BUFSIŻ	stdio. h	ファイル入出力バッファのバイト数

2.2 型とマクロの定義

マクロ	ファイル	内　　容
_NFILE	stdio. h	ファイル入出力用の構造体の配列の要素数 標準で5つのストリームがオープンされているのでユーザーが同時にオープンできるストリームの個数はこの値から5を引いた個数になります
FOPEN_MAX	stdio. h	ファイル入出力用の構造体の配列の票素数 標準で5つのストリームがオープンされているのでユーザーが同時にオープンできるストリームの個数はこの値から5を引いた個教になります
TMP_MAX	stdio. h	tmpfile関数などで作成できるテンポラリファイルの最大数
L_tmpnam	stdio. h	tmpname関数などで作成されるファイル名の長さ
FILENAME_MAX	stdio. h	ドライブ名、パス名などを付加したファイル名の最大

2.3 グローバル変数

インクルードファイルには、グローバル変数と呼ばれる、各プログラムから共通に使用される変数が外部宣言されています。
グローバル変数の種類とその内容は以下の通りです。

グローバル変数	ファイル	内　　容
time_t timezone	time. h	グリニッジ標準時と地方時の時差が秒数で格納されている 日本では−32400（−9時間）になっている
int daylight	time. h	夏時間を採用している地域では非0が格納されている 日本では0が格納されている
char ＊ tzname［2］	time. h	時刻地域名と夏時間名を示す文字列へのポインタの配列 文字列の長さは3文字 日本ではtzname［0］は"JST"、tzname［1］は""が定義されている
char tzstn［］	time. h	時刻地域名が格納されている
char tzdtn［］	time. h	夏時間名が格納されている
unsigned char ＊＊ environ	stdlib. h	環境文字列の配列へのポインタが格納されている
int _fmode	stdlib. h	ファイルの変換モードのディフォルト値が格納されている
unsigned int _PSP	stdlib. h	自分自身のプロセス管理ポインタの先頭のアドレスが格納されている
volatile int errno	stdlib. h	エラーの種類を表す数値が格納されている
int sys_nerr	stdlib. h	sys_errlist配列の要素数が格納されている
char ＊sys_errlist［］	stdlib. h	エラーメッセージ文字列の配列
volatile int _doserrno	stdlib. h	システムコールでエラーが発生した場合にそのエラーの番号が格納される

2.4 標準型構造体名

インクルードファイルでは標準型と呼ばれる構造体を宣言しています。
これらの構造体名とその説明を以下に示します。

標準型	内　容
complex	complex構造体はmath. hで宣言してあり、複素数の実数部と虚数部を格納する
exception	exception構造体はmath. hで宣言してあり、数値演算ルーチンのエラー情報を格納する
stat	stat構造体はstat. hで宣言してあり、statまたはfstat関数が返すファイルの詳細情報を格納する
tm	tm構造体はtime. hで宣言してあり、asctime、gmtime、localtimeの関数が時間の情報を読み書きするために使用する
utimbuf	utimbuf構造体はutime. hで宣言してあり、utime関数でファイルを修正した日付を変更するために使用する
timeb	timeb構造体はtimeb. hで宣言してあり、ftime関数で使用する

2.5 インクルードファイルの種類

XCで提供しているインクルードファイルには以下のものがあります。

ファイル名	内　　容
assert. h	実行時チェックマクロの定義
audio. h	ADPCMの制御関連の関数の宣言 (BASIC→Cの変換時に使用)
basic. h	X-BASIC本体組み込みの関数の宣言 (BASIC→Cの変換時に使用)
basic0. h	画面処理関係他の関数の宣言 (BASIC→Cの変換時に使用)
class. h	データ型式の定義
conio. h	コンソール入出力を行う関数の宣言
ctype. h	文字チェックおよび文字変換のマクロ定義
direct. h	ディレクトリ操作の関数宣言
doslib. h	Human68kのファンクションコールの関数宣言
error. h	エラーコードの宣言
fcntl. h	低水準入出力関数用定数の宣言
fctype. h	ctype. hで定義されているマクロの関数版の宣言
float. h	(算術ライブラリの)浮動小数点演算用定数の宣言
graph. h	グラフィック制御関連の関数の宣言 (BASIC→Cの変換時に使用)
image. h	イメージ制御関連の関数の宣言 (BASIC→Cの変換時に使用)
io. h	ファイル操作と低水準入出力関数の宣言
iocslib. h	X68000のIOCSコールの関数宣言

2.5 インクルードファイルの種類

ファイル名	内　　容
jfctype. h jstring. h	日本語ライブラリ用関数の宣言
limits. h	各データ形式のリミットチェック
math. h	算術ライブラリ用関数の宣言
mouse. h	マウスの制御関連の関数の宣言 (BASIC→Cの変換時に使用)
music. h	FM音源の制御関連の関数の宣言 (BASIC→Cの変換時に使用)
music2. h	MIDI対応FM音源の制御関連の関数の宣言 (BASIC→Cの変換時に使用)
process. h	プロセス制御関数の宣言
setjmp. h	setjmp関数用構造体などの宣言
signal. h	signal関数用定数などの宣言
sprite. h	スプライトの制御関連の関数の宣言 (BASIC→Cの変換時に使用)
stat. h	statおよびfstat関数用定数などの宣言
stdarg. h	可変引数をあつかうためのマクロ定義
stddef. h	汎用データ型、汎用マクロ定義
stdio. h	標準入出力関数などの宣言
stdlib. h	標準ライブラリ関数の宣言
stick. h	ジョイスティック入力用の関数の宣言 (BASIC→Cの変換時に使用)
string. h	文字列関数用文字列操作関数の宣言
time. h	time関数用構造体などの宣言
timeb. h	ftime関数用構造体などの宣言
utime. h	utime関数用構造体などの宣言

2.5.1 assert. h

論理エラーのチェックを行うassertマクロを定義します。
assertを使うときは必ずインクルードします。
もし、NDEBUGが定義されていないときassertにはassert関数がマクロ展開されます。
NDEBUGが定義されますとassertは空となります。
すなわち、ソースファイルからassertを取り除いたのと同じになります。

2.5.2 audio. h

ADPCMの録音および再生の機能を実行する関数を宣言しています。
なお、このインクルードファイルはBASIC→Cの変換時に使用します。

2.5.3 basic. h

X-BASIC本体組み込みの関数の宣言をしています。
おもな内容はファイル処理、数値演算、文字列処理などです。
なお、このインクルードファイルはBASIC→Cの変換時に使用します。

2.5.4 basic0. h

X-BASICの画面処理関係他の関数の宣言をしています。
主な内容は画面のクリア、画面サイズの設定、キー入力などです。
なお、このインクルードファイルはBASIC→Cの変換時に使用します。

2.5.5 class. h

データ形式（char、int等）を示すシンボルを定義しています。
このシンボルを使うことにより、他のCコンパイラで処理系によって異なる標準の型の違い（int語長など）を吸収し、移植性のよいプログラムが作成できます。

2.5.6 conio. h

コンソール入出力を行う関数を宣言しています。

2.5.7 ctype. h

文字チェックや文字の変換を行うマクロ、およびこれらのマクロで使用するフラグを定義しています。

2.5.8 direct. h

ディレクトリ操作を行う関数を宣言しています。

2.5.9 doslib. h

Human68kのファンクションコールの関数の宣言をしています。
これらの関数は、Human68kのファンクションコールを呼び出すもので、Cのライブラリで下から2番目（IOCSの上）に位置する関数です。
よって、上位の関数と組み合わせて使う場合、注意しないと上位の関数の動作がおかしくなるおそれがあります。

2.5.10 error. h

エラーナンバーの定義。
ANSI規格のerrno. hと同じ意味で使用されます。

2.5.11 fcntl. h

低水準入出力関数（open、close等）を宣言するとともに、これらの関数を呼び出すときに使用するフラグや、ファイル種別、マクロなどを定義しています。

2.5.12 fctype. h

ctype. hで定義しているマクロを、関数として宣言しています。

2.5.13 float. h

浮動小数点データの範囲などを定義しています。

2.5.14 graph. h

グラフィック制御で使用する関数を宣言しています。
このインクルードファイルは、BASIC→Cの変換時に使用します。

2.5.15 image. h

イメージ処理を行うときに必要となる関数を宣言しています。
なお、このインクルードファイルは、BASIC→Cへの変換時に使用します。

2.5.16 io. h

低水準入出力操作の関数を宣言しています。

2.5.17 iocslib. h

X68000のIOCSコールの関数を宣言しています。
C言語でこれらの関数を呼び出すことにより、X68000のハードウェアを、直接制御することができます。
これらの関数は、Cライブラリでもっとも下位に位置する関数ですので上位の関数と組み合わせて使う場合、注意しないと上位の関数の動作がおかしくなるおそれがあります。

2.5.18 jfctype. h

日本語対応版文字分類関数、および文字変換関数を宣言しています。

2.5.19 jstring. h

漢字を含む文字列を操作する関数を宣言しています。

2.5.20 limits. h

整数型と文字型のデータの範囲チェックを行うために、これらの型の最大値と最小値を定義しています。

2.5.21 math. h

数学ライブラリを宣言するとともに、マクロおよび、これらのライブラリで使用するシンボルを定義しています。
定義されたシンボルは変更してはいけません。

2.5.22 mouse. h

マウスを使用するときに必要となる関数を宣言しています。
なお、このインクルードファイルは、BASIC→Cの変換時に使用します。

2.5.23 music. h

FM音源の制御に際して必要となる関数を宣言しています。
なお、このインクルードファイルは、BASIC→Cの変換時に使用します。

2.5.24 music2. h

MIDI対応FM音源の制御に際して必要となる関数を宣言しています。
なお、このインクルードファイルは、BASIC→Cの変換時に使用します。

2.5.25 process. h

プロセス制御関数（signalを除く）を宣言しています。

2.5.26 setjmp. h

setjmp関数を使用するときに必要な構造体の定義や関数の宣言をしています。

2.5.27 signal. h

signal関数を使用するときに必要なシンボルを定義しています。

2.5.28 sprite. h

スプライト制御において使用する関数を宣言しています。
なお、このインクルードファイルは、BASIC→Cへの変換時に使用します。

2.5.29 stat. h

stat、fstat関数で返される構造体の定義、ファイルのステータス情報のフラグ定義、およびstat、fstatの関数宣言をしています。

2.5.30 stdarg. h

printf関数などのように、引数の数が不定個の関数を作成したときに、引数の取り出しを行うためのマクロ（va_start、va_arg、va_endなど）を定義しています。

2.5.31 stddef. h

頻繁に使用される値や、動作環境や機種で異なる部分を吸収するためのシンボルなどを定義しています。

ここで定義されたシンボルを使うことにより、移植性のよいプログラムが作成できます。

EOF

ファイルの終わり、もしくはエラーが見つかったときこの値が返されます。

NULL

ヌルポインタの値を表します。

2.5.32 stdio. h

ストリーム入出力のさいに使用する構造体（FILE）、シンボル（NULL、EOFなど）、マクロ（getchar、putchar、ferrorなど）、バッファサイズなどを定義しています。

stdio. hファイル中で定義している定数は、各プログラム中で共通に使用できますが、定義された値を変更してはいけません。

以下に、よく使われる定数の名前とその内容を説明します。

BUFSIZ

ストリームI／Oで使用するバッファのサイズを表します。
この値は、システムがバッファを確保するとき、もしくはsetbuf関数で別の

バッファを確保するときに参照されます。

_NFILE
1度にオープンできるファイルの数を表します。
stdin、stdout、stderr、stdaux、stdprnの5個のファイルはつねにオープンされています。

FILE構造体は、stdio. hで定義されています。
ストリームルーチンは、ストリームをアクセスするのにFILE構造体のポインタを使います。

2.5.33 stdlib. h

標準ライブラリを宣言しています。
このインクルードファイルに宣言されているライブラリを機能別に分けると以下のようになります。

- メモリ割りあて
- ソート機能
- データ変換
- その他

2.5.34 stick. h

ジョイスティック入力時に必要となる関数(stick、strig)を宣言しています。

2.5.35 string. h

文字列操作関数やメモリ操作関数を宣言するとともに、これらの関数を使用するときに必要となるシンボルを定義しています。

2.5.36 time. h

time関数を宣言するとともに、これらの関数を使用するときに必要となる構造体やシンボルを宣言しています。

2.5.37 timeb. h

ftime関数を宣言するとともに、timeb構造体を宣言しています。

2.5.38 utime. h

utime関数を定義するとともに、utimbuf構造体を宣言しています。

第3章

項目別ランタイムルーチン

バッファ操作/メモリ割りあて/文字の分類と変換
データ変換/ディレクトリ制御
ファイル操作/ストリーム入出力
低水準入出力/コンソール入出力
数値演算/プロセス制御
サーチとソート/文字列操作
時　間/日本語ライブラリ
BASIC ライブラリ/IOCS コール
DOS コール/その他

本書では、項目別にライタイムルーチンについて説明します。
BASICライブラリ、IOCSコール、DOSコールなどは、関数名とその機能を表組みで分かりやすく解説します。

3.1 バッファ操作

バッファ操作関数は、バッファと呼ばれる特定のメモリブロック、またはメモリブロック間において、文字の検索、データのコピー・比較などを行います。バッファ操作関数の宣言は、インクルードファイルstring.hで行われています。この関数には、バッファ操作関数と拡張バッファ操作関数の2種類があり、バッファ操作関数は他のCコンパイラでも使用できる標準的な関数であり、拡張バッファ操作関数はXCで拡張された関数です。
内訳はそれぞれ以下の通りです。

バッファ操作関数

関数名	機　能
memchr	指定されたバイト数のバッファ内から、指定された文字を検索する
memcmp	2つのバッファを先頭から指定文字数分比較する
memcpy	あるバッファから他のバッファへ、指定した数の文字列をコピーする
memmove	あるバッファから他のバッファへ、指定した文字数分をコピーする
memset	バッファの先頭から指定バイト数分、指定した文字で初期化する

拡張バッファ操作

関数名	機　能
memccpy	指定された文字が現れるまでバッファ内の文字列を他のバッファにコピーする
memcmpi	2つのバッファを大文字／小文字の区別なしで先頭から指定文字数分比較する
memicmp	2つのバッファを大文字／小文字の区別なしで先頭から指定文字数分比較する

3.1 バッファ操作

関数名	機　能
movedata	あるバッファから他のバッファへ、指定した文字数分をコピーする
movmem	あるバッファから他のバッファへ、指定した文字数分をコピーする
repmem	メモリブロック中の指定箇所を複数回コピーする
setmem	バッファの先頭から指定バイト数分、指定した文字で初期化する
swmem	2つのメモリブロックの内容を入れ替える

3.2 メモリ割りあて

メモリ割りあて関数は、メモリ領域の確保、解放、再割りあてを行います。なお、これらの関数のうちsbrkは低水準メモリ割りあてルーチンで、割りあてられていないメモリ領域を利用可能とします。
メモリ割りあて関数には以下のものがあり、これらの関数はすべてインクルードファイルstdlib. hで宣言されています。

関数名	機　能
allmem	使用可能なすべてのメモリを確保する
bldmem	メモリブロックをキロバイト単位で確保する
calloc	配列の領域を割りあてる
chkml	未使用のメモリブロックのうち、最大のもののサイズをバイトで通知する
free	calloc、malloc、reallocで割りあて済みのメモリブロックを解放する
getmem	指定されたサイズのメモリ領域を確保する
getml	指定されたサイズのメモリ領域を確保する
malloc	メモリブロックを割りあてる
rbrk	ブレーク値を初期状態に戻す
realloc	割りあて済みのメモリブロックのサイズを変更する
rlsmem	getmemで確保したメモリ領域を解放する
rlsml	getmlで確保したメモリ領域を解放する
rstmem	ブレーク値を初期状態に戻す

3.2 メモリ割りあて

関数名	機　　能
sbrk	ブレーク値を変更する（ヒープ領域を拡張する）
sizmem	使用可能なメモリブロックのサイズをワードで通知する
swab	ワードデータ列を上位／下位を入れ替えて転送する
swaw	ロングワードデータを上位／下位を入れ替えて転送する

注：sbrk 関数は他のメモリ割りあて関数と一緒に使うべきものではありません。
特に、sbrk でサイズを小さくし、他のメモリ割りあて関数を使用した場合、動作はまったく保証されません。
また、rbrk 関数も他のメモリ割りあて関数と一緒に使ってはいけません。

3.3 文字の分類と変換

これらは、文字の分類（判定）や、大文字～小文字間の変換などを行うマクロ（または関数）であり、すべてインクルードファイルctype. hとfctype. hで宣言されています。
これらのマクロ（または関数）には以下のものがあります。

関数名	機　能
isalnum	英数字の判定を行う
isalpha	英文字の判定を行う
isascii	ASCII 文字の判定を行う
iscntrl	制御文字の判定を行う
iscsym	C 言語の識別子として使用できる文字の判定を行う
iscsymf	C 言語の識別子の先頭文字として使用できる文字の判定を行う
isdigit	10 進数の判定を行う
isgraph	空白以外の表示可能文字の判定を行う
islower	英小文字の判定を行う
isodigit	8 進数の判定を行う
isprint	表示可能文字の判定を行う
ispunct	句読点の判定を行う
isspace	空白文字の判定を行う
isupper	英大文字の判定を行う
isxdigit	16 進数の判定を行う
toascii	与えられた文字を ASCII コードへ変換する
tolower	文字を判定し、大文字ならば小文字へ変換する

3.3　文字の分類と変換

関数名	機　能
toupper	文字を判定し、小文字ならば大文字へ変換する
_tolower	大文字を小文字へ変換する(A～Z 以外は不定)
_toupper	小文字を大文字へ変換する(a～z 以外は不定)

これらは、マクロと関数の両方が用意されています。
ctype. h をインクルードするとマクロが、fctype. h をインクルードすると関数が使用されます。

これらのうち、isodigit、iscsym、iscsymf、tolower、toupper を関数として使用する場合は良いのですが、マクロとして使用する際には、注意が必要となります。
以上の5つは、マクロで使用した場合に引数を複数回評価しますので、副作用を伴う引数を指定した場合のみ、結果が正しく得られない場合があります。

他の文字分類、文字変換を関数として呼び出しても良い場合には、fctype. h をインクルードして、すべてを関数として宣言してください。

副作用を伴わない引数を指定しているものはマクロとして、副作用を伴う引数を指定しているものは関数として宣言したい場合は、以下のようにしてください。

#include <fctype. h> ……副作用のあるものを関数として宣言するため fctype. h をインクルードする。
#include <ctype. h> ……一旦、すべてをマクロとして宣言する。
#undef tolower……………関数として呼び出すもののみマクロの宣言を無効にする。

この方法のように ctype. h と fctype. h の両方をインクルードする場合は、必ず fctype. h を先にインクルードしてください。
もし、ctype. h をインクルードした後で fctype. h をインクルードすると、fctype. h は無効になるので注意してください。

3.4 データ変換

データ変換関数は、数値～文字列間の変換を行う関数であり、以下のような場合に使用します。

- 文字列を、コンピュータ内部で演算可能な表現形式に変換したいとき
- コンピュータ内部の数値表現形式を、画面に表示可能な、またはプリンタに印字可能な文字列に変換したいとき

データ変換関数には以下のものがあり、すべてインクルードファイルstdlib.hで宣言されています。

関数名	機　　能
atof	数値を表す文字列を倍精度の浮動小数点値に変換する
atoi	数値を表す文字列を整数値に変換する
atol	数値を表す文字列をlong型整数値に変換する
atow	数値を表す文字列をshort型整数値に変換する
ecvt	倍精度の浮動小数点値を数値を表す文字列に変換する
fcvt	倍精度の浮動小数点値を数値を表す文字列に変換する
gcvt	倍精度の浮動小数点値を数値を表す文字列に変換する
itoa	整数を文字列に変換する
ltoa	long型整数を文字列に変換する
strtod	数値を表す文字列を浮動小数点値に変換する
strtol	符号が付いても良い数値を表す文字列をlong型整数値に変換する
strtoul	符号が付かない数値を表す文字列をunsigned long型整数値に変換する
uitoa	符号なし整数を文字列に変換する

3.4　データ変換

関数名	機　　能
ultoa	符号なし long 型整数を文字列に変換する
uwtoa	符号なし short 型整数を文字列に変換する
wtoa	short 型整数を文字列に変換する

3.5 ディレクトリ制御

ディレクトリ制御関数は、ディレクトリに関する操作を行います。
ディレクトリ制御関数には以下のものがあり、すべてインクルードファイルdirect. hで宣言されています。

関数名	機　　能
chdir	現在のワーキングディレクトリを変更する
getcwd	現在のワーキングディレクトリを通知する
mkdir	ディレクトリを作成する
rmdir	ディレクトリを削除する

3.6 ファイル操作

ファイル操作関数は、パス名やファイルハンドルで指定したファイルの管理情報の通知と変更、ファイルサイズの通知と変更、ファイル名の変更、ファイルの削除などを行います。
これらの関数には以下のものがあり、fstat、stat関数はインクルードファイルstat. hで、これ以外の関数はインクルードファイルio. hで宣言されています。

関数名	機　　能
access	ファイルの許可設定を調べる
chgfa	ファイルの属性を変更する
getfa	ファイルの属性を調べる
chgft	ファイルの作成日時を変更する
getft	ファイルの作成日時を調べる
dtasc	ファイル情報の日付から文字列を作る
dtcnv	文字列からファイル情報用の日付を作る
tmasc	ファイル情報の時刻から文字列を作る
tmcnv	文字列からファイル情報用の時刻を作る
chmod	ファイルの許可設定を変更する
chsize	ファイルの大きさを変更する
filelength	ファイルの長さを調べる
files	ファイルを検索しファイルの情報を得る
nfiles	ファイルを検索しファイルの情報を得る
fstat	ファイルハンドルのステータス情報を得る
ftpack	年月日時分秒をファイル情報用の日時にする
ftunpk	ファイル情報用の日時を年月日時分秒にする

関数名	機　　能
isatty	文字デバイスであるかどうかを調べる
mktemp	仮のファイル名を作る
rename	ファイル名を変更する
setmode	ファイルの変換モードを設定する
stat	ファイル名からファイルのステータス情報を得る
unlink	ファイルを削除する

以下の関数を使用するときは、ファイルはパス名またはファイル名によって指定します。

access
chgfa
chmode
files
getfa
nfiles
rename
remove
stat
unlink
utime

以下の関数では、ファイルの指定はファイルハンドルで行います。

chgft
chsize
filelength
fstat
getft
isatty
setmode

3.7 ストリーム入出力

ストリーム入出力関数は、C標準ライブラリの入出力ルーチンのうちの1つですが、他の入出力ルーチンと比べた場合、以下のような特徴があります。

- データファイルを独立した文字のストリームとして扱う
- データのバッファリングが可能となる
- データをフォーマット化して入出力を行うことができる
- ファイルの管理はFILE構造体によって行われる

これらの詳細については、この後で述べる各項目別の基本事項の中でふれます。

ストリーム入出力関数には以下のものがあり、これらの関数およびこれらの関数を使用するにあたって必要とする定数、型、構造体などは、すべてインクルードファイルstdio. hで定義されています。

関数名	機　能
clearerr (clrerr)	ストリームのためのエラーインジケータをクリアする
fclose	ストリームをクローズする
fcloseall	オープンされいるすべてのストリームをクローズする
fdopen	ファイルハンドルを使ってストリームをオープンする
feof	指定されたストリームがエンドオブファイルか否か調べる（マクロ）
ferror	ストリームの入出力時のエラーを調べる（マクロ）
fflush	ストリームをフラッシュする

3.7　ストリーム入出力

関数名	機　能
fgetc	ストリームから1文字読み込む（関数）
fgetchar	標準入力から1文字読み込む（関数）
fgetpos	現在のファイルポインタの位置を得る
fsetpos	ファイルポインタを所定の位置へ移動する
fgets	ストリームから文字列を読み込む
fileno	ストリームのためのファイルハンドルを得る（マクロ）
flushall	すべてのストリームをフラッシュする
fmode	ファイルの変換モードを変更する
fopen	ストリームをオープンする
fprintf	データをフォーマットしてストリームに書き込む
fputc	ストリームへ1文字書き込む（関数）
fputchar	標準出力へ1文字書き込む（関数）
fputs	ストリームへ文字列を書き込む
fread	ストリームから固定長データ項目を読み込む
freopen	ストリームポインタを再割りつけする
fscanf	ストリームからデータをフォーマットにしたがって読み込む
fseek	ファイルポインタを所定の位置へ移動する
ftell	現在のファイルポインタの位置を得る
fwrite	ストリームへ固定長データ項目を書き込む
getc	ストリームから1文字読み込む（マクロ）
getchar	標準入力から1文字読み込む（マクロ）
getl	ストリームからint型のデータを読み込む
gets	標準入力から1行読み込む

3.7　ストリーム入出力

関数名	機　能
getw	ストリームから short 型のデータを読み込む
printf	標準出力へデータをフォーマットして書き込む
putc	ストリームへ1文字書き込む（マクロ）
putchar	標準出力へ1文字書き込む（マクロ）
putl	ストリームへ int 型のデータを書き込む
puts	ストリームへ1行書き込む
putw	ストリームへ short 型のデータを書き込む
remove	ファイルを削除する
rewind	ファイルポインタをファイルの先頭に移動する
rmtemp	すべての作業用ファイルを削除する
rmtmp	すべての作業用ファイルを削除する
tempfile	作業用ファイルを新規に作成する
tmpfile	作業用ファイルを新規に作成する
tempnam	作業用ファイル用のファイル名を作る
tmpnam	作業用ファイル用のファイル名を作る
scanf	標準入力からデータをフォーマットして読み込む
setbuf	ストリームバッファの制御を行う
setnbf	ストリームバッファを変更する
setvbuf	ストリームバッファを変更する
sprintf	文字列へデータをフォーマットして書き込む
sscanf	文字列からデータをフォーマットして読み込む
ungetc	読み込んだ文字を入力ストリームに返す
vfprintf	データをフォーマットしてストリームに書き込む
vprintf	標準出力へデータをフォーマットして書き込む
vsprintf	文字列へデータをフォーマットして書き込む

3.7　ストリーム入出力

注：ストリーム入出力関数と低水準入出力関数は一般に互換性がないので1個のファイルに対して両者を併用してはなりません。

たとえば、ストリーム入出力関数はバッファリングを行うのに対して、低水準入出力関数はこれを行わないので、両者を併用してアクセスすると、バッファ内のデータが失われることがあります。

getc、putc、getchar、putchar、feof、ferror、fileno、clearerr の8つの関数は、マクロと関数の両方が用意されています。
マクロを使用する場合には、stdio. h をインクルードする前に以下のようにしてください。

```
#define MACRO
```

関数を使用する場合には、stdio. h をインクルードする前に以下のようにしてください。

```
#undef MACRO
```

ストリームのオープン
ストリームの入出力を行うに先立って、ストリームをオープンしなければなりません。
ストリームのオープンは、fopen、fdopen、freopen 関数によって行います。これらの関数は、FILE 構造体と呼ばれる、ファイルの管理に必要な情報を格納している構造体へのポインタを返してきますので、以後このポインタを使ってストリームへのアクセスが行えるようになります。

定義済みのストリームポインタ
ストリームポインタの中には、自動的にオープンされるため、上記の関数によるオープン処理を必要としないものもあります。
このようなストリームポインタを以下の表に示します。

ストリーム	対応する入出力デバイス
stdin	標準入力（キーボード）
stdout	標準出力（画面）
stderr	標準エラー出力（画面）
stdaux	標準補助入出力（RS-232C）
stdprn	標準プリンタ出力（プリンタ）

なお、標準入出力の入力先、および出力先は、入出力リダイレクション機能により変更することができます。
ただし、入出力リダイレクトされているときにfcloseやfcloseall関数などによってクローズしてしまうと、入出力リダイレクトは無効になってしまうので注意してください。
詳細については、「Cユーザーマニュアル」を参照してください。

ストリームバッファの制御

ストリーム入出力関数によってオープンしたファイルは、自動的にバッファリング、すなわちバッファを確保し、このバッファを経由して入出力が行われるようになります。
ただし、オープン済みのストリームstdinやstdoutなどはこのかぎりではありません。
これらのストリームはラインバッファ化され、改行を入力するとフラッシュされます。
setbuf関数によって、以下のような、バッファに対する特殊な制御を行うことができます。

- ●ストリームのバッファリングを抑止する
- ●バッファリングされていないストリームとバッファを結合する
- ●（setbuf関数によって割りつけられた）システムバッファをユーザーが命名し、アクセスできるようにする

バッファサイズは、インクルードファイルstdio. hの中で、記号定数BUF_SIZで定義されおり、バッファのアクセスに際しては、このサイズを使用します。

3.7　ストリーム入出力

プログラムでストリームからの入力を行うと、データはいったんバッファへ読み込まれ、次にユーザー領域へ転送されます。
次にストリームの読み込み要求を行うと、データはファイルデバイスからではなく、このバッファから読み込まれます。
バッファからすべてのデータが読み込まれると、再びファイルデバイスからデータがバッファに読み込まれます。
これに対して、ストリームの出力を行うと、データは次々に出力バッファへ書き込まれます。
そして出力バッファがフルになると、自動的にファイルデバイスへの書き出しが行われます。
また、出力バッファがフルになっていなくても、ファイルがクローズされると、自動的に書き出しが行われます。
これを、バッファのフラッシュといいます。
上記で述べたように、通常、バッファのフラッシュは自動的に行われますが、fflush、flushall 関数により、強制的にフラッシュすることもできます。

ファイルポインタによるストリームのアクセス

ストリームへのアクセス、すなわちストリームのデータの読み込み、およびストリームへのデータの書き込みは、ファイルポインタにより行います。
ファイルポインタは、通常、システムによって設定と更新が行われます。
たとえば、

- ストリームを読み込みモードでオープンしたときには、ファイルポインタはファイルの先頭に設定され、1文字読み込むたびに、次に読み込むべき位置を指すように更新される。

- ストリームを書き込みモードでオープンしたときには、ファイルポインタはファイルの先頭に設定され、1文字書き込むたびに、次に書き込むべき位置を指すように更新される。

- ストリームを追加（アペンド）モードでオープンしたときには、ファイルポインタはファイル内の最終データの次の位置に決定され、1文字書き込むたびに、次に書き込むべき位置を指すように更新される

これに対して、複雑なファイル処理が必要となり、ユーザー自身がファイルポインタを操作するために、以下の関数が用意されています。

fseek ………ファイルポインタを任意の位置に設定する
rewind ……ファイルポインタをファイルの先頭に設定する
ftell…………ファイルポインタの現在の位置を通知する
feof…………ファイルポインタの現在の位置がエンドオブファイルであるか否か通知する

注：fseek 関数を使用する場合、テキストモードでオープンしたファイルのファイルポインタを変更する場合には、以下のいずれかの条件に合わない位置に変更してはいけません。

- ファイルの先頭の位置
- ftell 関数で返された位置
- ファイルの最終データの次の位置

エラーの発見

ストリームの入出力によってエラーが発生した場合、ストリーム上のエラーインジケータがセットされます。
エラーが発生したか否かは、ferror 関数により判定することができます。

エラーインジケータがセットされると、以下の処理を行わないかぎりエラーインジケータは現在の状態を保存します。

- ストリームをクローズする
- rewind 関数により、ファイルをリワインドする
- clearerr 関数により、エラーインジケータをクリアする

ストリームのクローズ

ファイルの処理が終了したら、fclose、fcloseall 関数により、ファイルをクローズします。
fcloseall 関数は、現在処理中のファイルだけでなく、オープンされているファイルをすべてクローズします。
なお、ストリーム stdin、stdout などはクローズする必要はありません。
プログラムが終了すれば、オープン済みのファイルはすべてクローズされますが、オープンできるファイルの数に限りがある場合もありますので、オープンしたファイルは必ずクローズするようにしましょう。

3.8 低水準入出力

低水準入出力関数は、オペレーティングシステムの入出力機能を直接呼び出すもので、他の入出力ルーチンと比較した場合、以下のような特徴があります。

- データのバッファリングとフォーマット化は行われない
- オープンされたファイルは、ファイルハンドルによって参照される

低水準入出力関数には以下のものがあり、これらの関数はすべてインクルードファイル io. hで宣言されています。

関数名	機　能
close	ファイルをクローズする
creat	ファイルを新規に作成する
dup	ファイルハンドルを複製する
dup2	ファイルハンドルを強制的に再割りつけする
eof	ファイルがエンドオブファイルであるか否か判定する
lseek	ファイルポインタを指定位置へ移動する
open	ファイルをオープンする
read	ファイルからデータを読み込む
tell	現在のファイルポインタの位置を得る
write	ファイルにデータを書き込む

ファイルのオープン

ファイルの入出力を行うに先立って、ファイルをオープンしておかなければなりません。

ファイルのオープンは、open、creat 関数によって行います。

これらの関数は、ファイルハンドル（ファイル記述子）を返してきますので、以後これを使ってファイルをアクセスすることができます。

ファイルは読み込み、書き込み、あるいはその両方のモードで、またテキストモードでもバイナリモードでもオープンすることができます。

定義済みのファイルハンドル

以下の表にある5つのファイルハンドルは自動的にオープンされるので、ファイルをオープンすることなく、低水準入出力関数から使用することができます。
また、ストリーム入出力関数と同様に、標準入出力のリダイレクションを行うことができます。

ハンドル	対応するストリーム	対応する入出力デバイス
0	stdin	標準入力（通常はキーボード）
1	stdout	標準出力（通常はスクリーン）
2	stderr	標準エラー出力（スクリーン）
3	stdaux	標準補助入出力（RS-232C）
4	stdprn	標準プリンタ出力（プリンタ）

データの読み出し／書き込み

データの入出力は、readおよびwriteという2つの基本的な関数により行います。
ファイルの読み出しと書き込みは、ストリーム入出力関数と同様に、ファイルポインタによって行います。
ファイルポインタは読み書きのつど、自動的に更新されますが、以下の関数により、ファイルポインタの操作を行うことができます。

lseek ………ファイルポインタを任意の位置に設定する
tell …………ファイルポインタの現在の位置を通知する
eof …………ファイルポインタの現在の位置がエンドオブファイルであるか否か通知する

注：lseek関数を使用する場合、テキストモードでオープンしたファイルのファイルポインタを変更する場合には、以下のいずれかの条件に合わない位置に変更してはいけません。

- ファイルの先頭の位置
- tell関数で返された位置
- ファイルの最終データの次の位置

ファイルのクローズ

ファイルのクローズは、close関数を使用して行います。

プログラムが終了すれば、オープンされていたファイルはすべて自動的にクローズされますが、オープンできるファイルの数が限られている場合もあるので、オープンしたファイルは必ずクローズするようにしましょう。

3.9 コンソール入出力

これらの関数は、コンソールと直接入出力を行うものですが、コマンド行で入出力のリダイレクションを行うことも可能です。
コンソール入出力関数には以下のものがあり、これらはすべてインクルードファイル conio. hで宣言されています。

関数名	機　　能
cgets	コンソールから文字列を読み込む
cprintf	コンソールへデータをフォーマットして出力する
cputs	コンソールへ文字列を出力する
cscanf	コンソールからデータをフォーマットして読み込む
getch	コンソールから1文字読み込む
getche	コンソールから1文字読み込み、エコーバックする
kbhit	キー入力の有無を調べる
putch	コンソールへ1文字出力する
ungetch	コンソールから読み込んだ文字を、次の読み込みのために返す
vcprintf	コンソールへデータをフォーマットして出力する

3.10 数値演算

これらの関数は、一般的な数値演算を行います。
数値演算関数では、浮動小数点を扱うので、浮動小数点サポートが必要となります。
数値演算関数には以下のものがあり、div、ldiv 関数がインクルードファイル stdlib. hで宣言されているほかは、すべて math. hで宣言されています。

関数名	機　能
acos	アークコサインを求める
asin	アークサインを求める
atan	アークタンジェントを求める
atan2	アークタンジェントを求める
cabs	複素数の絶対値を求める
ceil	Xと等しいかXより大きい最小の整数値を求める
cos	コサインを求める
cosh	ハイパボリックコサインを求める
div	int の割り算の商と余りを得る
except	数値演算ライブラリ使用時の例外処理を行う
exp	指数関数を求める
fabs	浮動小数点値の絶対値を求める
floor	Xと等しいかXより小さい最大の整数値を求める
fmod	剰余を求める
frexp	浮動小数点値を仮数部と指数部に分ける
hypot	直角三角形の斜辺の長さを求める
ldexp	Xと2のn乗の積を求める

3.10 数値演算

関数名	機　　能
ldiv	long の割り算の商と余りを得る
log	自然対数を求める
log10	10 を底とした対数を求める
matherr	数値演算関数のエラー処理を行う
modf	浮動小数点値を整数部と分数部に分ける
pow	X^Yを求める
sin	サインを求める
sinh	ハイパボリックサインを求める
sqrt	平方根を求める
tan	タンジェントを求める
tanh	ハイパボリックタンジェントを求める

3.11 プロセス制御

これらの関数はプロセスの制御を行います。
「プロセス」とは、オペレーティングシステムにおける実行の単位であり、プログラムのコード、データ、資源、およびこれらに関する制御情報などから成ります。
プロセス制御関数は、具体的には、以下のことを行います。

- プロセスを終了させる
- シグナル割り込みの制御を行う
- 新しいプロセスを起動する

プロセス制御関数には以下のものがあり、signal、raise 関数はインクルードファイル signal. h で宣言されています。
また、abort、atexit、exit、_exit、system の各関数は、インクルードファイル stdlib. h で宣言されています。
これ以外の関数はインクルードファイル process. h で宣言されています。

関数名	機　能
abort	プロセスを異常終了する
atexit	exit 時に呼び出す関数を登録する
execl	引数リストによって子プロセスを実行する
execle	引数リストと与えられた環境により子プロセスを実行する
execlp	引数リストと path 変数によって子プロセスを実行する
execlpe	引数リスト、path 変数、および引数 envp により子プロセスを実行する
execv	引数配列によって子プロセスを実行する

3.11 プロセス制御

関数名	機　能
execve	引数配列と与えられた環境によって子プロセスを実行する
execvp	引数配列と path 変数によって子プロセスを実行する
execvpe	引数配列、path 変数、および引数 envp により子プロセスを実行する
exit	プロセスを終了する
_exit	バッファをフラッシュせずに子プロセスを終了する
getpid	プロセスの ID を得る
raise	プログラムにシグナルを送る
signal	シグナル割り込みのための操作を行う
spawnl	引数リストによって子プロセスを実行する
spawnle	引数リストと与えられた環境により子プロセスを実行する
spawnlp	引数リストと path 変数によって子プロセスを実行する
spawnv	引数配列によって子プロセスを実行する
spawnve	引数配列と与えられた環境によって子プロセスを実行する
spawnvp	引数配列と path 変数によって子プロセスを実行する
system	Human68k コマンドを実行する
wait	起動した子プロセスの終了コードを得る

これらの関数では、子プロセスを起動するための path 変数の設定の有無、子プロセスへの引数のわたしかた、子プロセスの実行環境（の設定方法）の違いにより、8つのパターンに分けられるので、これを次頁の表に示します。

関数名	path 変数の設定	引数のわたしかた	子プロセスの実行環境
execl spawnl	path は使用しない	引数リスト	親プロセスの環境を引き継ぐ
execle spawnle	path は使用しない	引数リスト	最後の引数として子プロセスの環境テーブルがわたされる
execlp spawnlp	path を使用する	引数リスト	親プロセスの環境を引き継ぐ
execv spawnv	path は使用しない	引数配列	親プロセスの環境を引き継ぐ
evecve spawnve	path は使用しない	引数配列	最後の引数として子プロセスの環境テーブルがわたされる
execvp spawnvp	path を使用する	引数配列	親プロセスの環境を引き継ぐ
execlpe	path を使用する	引数リスト	最後の引数として子プロセスの環境テーブルがわたされる
execvpe	path を使用する	引数配列	最後の引数として子プロセスの環境テーブルがわたされる

3.12 サーチとソート

これらの関数は、インクルードファイル stdlib. hで宣言されています。

関数名	機　能
bsearch	バイナリサーチを実行する
dqsort	double 型のデータをソートする
fqsort	float 型のデータをソートする
lqsort	long 型のデータをソートする
qsort	クイックソートを実行する
sqsort	short 型のデータをソートする
tqsort	ポインタ型のデータをソートする

3.13 文字列操作

これらの関数は、ヌル文字で終了する文字列の操作を行います。
ヌル文字で終了しない文字列については、バッファ操作関数を使用してください。
文字列操作関数には、以下のものがあり、これらはすべてインクルードファイル string. h で宣言されています。

関数名	機　能
strcat	文字列を追加する
strchr	文字列において、指定された文字の最初の位置を調べる
strcmp	2 つの文字列を比較する
strcmpi	2 つの文字列を大文字／小文字の区別なしで比較する
strcpy	文字列を指定された場所へコピーする
strcspn	文字列において、指定文字セットの最初の文字を探す
strdup	文字列を複写する
stricmp	2 つの文字列を大文字／小文字の区別なしで比較する
strlen	文字列の長さを調べる
strlwr	文字列中の大文字をすべて小文字に変換する
strncat	文字列の先頭から n 文字分、他の文字列に追加する
strncmp	先頭から n 文字分、2 つの文字列を比較する
strncpy	文字列中の n 文字を他の文字列にコピーする

3.13 文字列操作

関数名	機　能
strnset	文字列の先頭n文字を、指定された文字で初期化する
strpbrk	他の文字列の構成文字を、指定文字列中で探す
strrchr	文字列中で、指定された文字が最後に現れる位置を探す
strrev	文字列の文字の並びを逆にする
strset	文字列の全文字を、指定された文字で置き換える
strsfn	ファイル名をドライブ名、パス名、主ファイル名、拡張子に分解する
strspn	文字列中から、指定文字セット以外の文字の位置を探す
strstr	文字列の中から指定した文字列と一致する部分を検索する
strtok	文字列の次のトークンを探す
strupr	文字列中の小文字をすべて大文字に変換する
strins	文字列を、他の文字列の前に挿入する
strmfe	ファイル名の拡張子を置き換える
strmfn	ドライブ名、パス名、主ファイル名、拡張子からファイル名を作成する
strmfp	パス名、主ファイル名からファイル名を作成する
stcgfe	ファイル名から拡張子をとり出す
strbpl	文字列の並びをもとに、ポインタ配列を作る
stcgfn	ファイル名から主ファイル名をとり出す
strsrt	文字列の配列をASCIIコード順にソートする

3.14 時間

時間関数は、システム時間から現在の時間を得るとともに、必要に応じてそれを変換し、格納します。

たとえば、time、ftime 関数は、グリニッジ標準時の 1970 年 1 月 1 日から現在まで経過した時間を秒単位で返します。
また、上記の関数で得た時間は、asctime、ctime、gmtime、localtime 関数を使用することにより、さまざまな型に変換することができます。

時間関数で使用するインクルードファイルは以下の通りです。

- ftime、utime以外 …… time. h（関数宣言）
- ftime …………………… timeb. h（関数宣言）
- utime …………………… utime. h（関数宣言）

ftime、localtime の地方時調整には、グローバル変数 daylight、timezone が参照されます。
これらは、日本での値で初期化されています。
上記の変数については、「第 2 章　2.3 グローバル変数」の daylight、timezone をそれぞれ参照してください。

関数名	機　　能
asctime	tm 構造体の時間情報を文字列に変換する
ctime	時間を time_t 型数値から文字列に変換する
ftime	現在のシステム時間を tm 構造体で得る
gmtime	時間を time_t 型数値から tm 構造体に変換する
time	現在のシステム時間を time_t 型数値で得る
tzset	環境時間変数から外部時間変数を設定する
clock	プログラムが起動してからの時間を得る

3.14 時間

関数名	機　　能
difftime	時間の差を浮動小数点値で得る
localtime	現在の地方時を得る
mktime	tm 構造体から time_t 型数値に変換する
strftime	tm 構造体で指定する日時を指定した書式で文字列に変換する
utime	ファイルの作成日時を変更する

3.15 日本語ライブラリ

日本語ライブラリには、日本語処理において入力される文字の分類を行う関数と、漢字処理関数があります。

3.15.1 文字分類関数

文字分類関数には以下のものがあり、すべてインクルードファイルjfctype. hで宣言されています。

関数名	機　能
iskana	カナ文字またはカナ句読点の判定を行う
iskpun	カナ句読点の判定を行う
iskmoji	カナ文字の判定を行う
isalkana	英文字またはカナ文字の判定を行う
ispnkana	英句読点またはカナ句読点の判定を行う
isalnmkana	英数字またはカナ文字の判定を行う
isprkana	表示可能文字（空白を含む）の判定を行う
isgrkana	表示可能文字（空白を除く）の判定を行う
iskanji	漢字の 1 バイト目の判定を行う
iskanji2	漢字の 2 バイト目の判定を行う

3.15.2 漢字処理関数

漢字処理関数には、漢字を含む文字列の操作を行う関数と、全角文字の分類、および全角文字～半角文字の変換を行う関数の 2 種類があります。

漢字を含む文字列の操作を行う関数には、以下のものがあります。

3.15 日本語ライブラリ

これらはすべて、インクルードファイルjstring. hで宣言されています。

関数名	機　能
jstrncmp	strncmp関数の漢字対応版で、漢字を含む2つの文字列を先頭からn文字分比較する
jstrchr	strchr関数の漢字対応版で、漢字を含む文字列において、指定文字が最初に現れる位置を調べる
jstrrchr	strrchr関数の漢字対応版で、漢字を含む文字列において、指定文字が最後に現れる位置を調べる
jstrcmp	strcmp関数の漢字対応版で、漢字を含む2つの文字列の比較を行う

全角文字の分類、および全角文字～半角文字の変換を行う関数には以下のものがあります。
これらはすべて、インクルードファイルjfctype. hで宣言されています。

関数名	機　能
jiszen	全角文字の判定を行う
jisl0	全角空白文字またはJIS第一水準の記号の判定を行う
jisl1	JIS第一水準漢字の判定を行う
jisl2	JIS第二水準漢字の判定を行う
jisalpha	全角英文字の判定を行う
jisupper	全角英大文字の判定を行う
jislower	全角英小文字の判定を行う
jisdigit	全角数字の判定を行う
jiskata	全角カタカナ文字の判定を行う

3.15　日本語ライブラリ

関数名	機　　能
jishira	全角ひらがな文字の判定を行う
jiskigou	全角句読点文字の判定を行う
jisspace	全角空白文字の判定を行う
hantozen	半角文字を全角文字に変換
zentohan	全角文字を半角文字に変換

3.16 BASICライブラリ

BASICライブラリは、BASTOC、すなわちBASICからCへの変換処理において使用します。

BASICライブラリには以下のものがあり、これらはすべてインクルードファイルaudio. h、basic. h、またはbasic0. h、graph. h、image. h、mouse. h、music. h、またはmusic2. h、sprite. h、stick. hで宣言されています。

関数名	機　能
a_play	PCM録音を再生する
a_rec	PCM録音を行う
a_cont	一時中断していたPCMの動作を再開する
a_end	PCMの動作を強制終了する
a_stat	PCMの動作状態を得る
a_stop	PCMの動作を一時中断する
asc	文字列の先頭の文字コードを返す
b_binS	整数を2進数の文字列に変換する
b_chrS	文字コードを文字に変換する
fix	浮動小数点数値の整数部を返す
b_hexS	整数を16進数の文字列に変換する
b_itoa	整数を文字列に変換する
b_octS	整数を8進数の文字列に変換する
val	文字列を数値に変換する
sgn	浮動小数点値の符号を得る
dskf	指定ドライブのディスクの空き容量を通知する
b_fclose	指定されたファイルをクローズする

3.16　BASICライブラリ

関数名	機　能
b_fcloseall	すべてのファイルをクローズする
b_feof	指定されたファイルがエンドオブファイルか否か調べる
b_fgetc	指定されたファイルから1文字読み込む
b_fopen	指定されたファイルをオープンする
b_fputc	指定されたファイルへ1文字書き出す
b_fread	指定されたファイルからデータを読み込む
b_freads	指定されたファイルからデータを読み込む
b_fseek	ファイルポインタの位置を移動する
b_fwrite	指定されたファイルにデータを書き出す
b_fwrites	指定されたファイルにデータを書き出す
randomize	rndの初期化を行う
b_int	浮動小数点を整数に変換する
rnd	浮動小数点数値の擬似乱数を発生させる
instr	文字列の中から特定の文字の並びを探し出す
b_leftS	文字列を切り出す
b_midS	文字列を切り出す
b_mirrorS	文字列の並びを反転する
b_rightS	文字列を切り出す
b_strchr	文字列中の最初にみつかった指定文字位置を返す
b_stringS	文字列中の1文字を複数個並べた文字列を返す
b_strrchr	文字列中の最後にみつかった指定文字位置を返す
b_strtok	指定文字を文字列から探す
b_dateS	日付を文字列に変換して通知する
b_timeS	時刻を文字列に変換して通知する

3.16 BASICライブラリ

関数名	機　　能
b_dayS	曜日を通知する
b_free	フリーエリアの大きさを調べる
b_inkeyS	キーの入力を待って読みとった1文字を返す
b_inkey0	キーの入力待ちをしないで読みとった1文字を返す
b_striS	整数値を文字列に変換する
b_strfS	浮動小数点数値を文字列に変換する
csrlin	カーソル位置のY座標を通知する
pos	カーソル位置のX座標を通知する
b_spaceS	スペースで埋まった文字列を作る
using	書式を指定して数値などを文字列に変換する
beep	ベルを鳴らす
cls	テキスト画面をクリアする
b_exit	プログラムを終了して親プロセスに戻る
child	子プロセスを起動する
width	画面の1行の表示文字数を設定する
color	テキスト画面の文字属性を設定する
console	スクロール行の範囲を設定する
b_input	キーから1行入力し、書式に従って変換し変数に代入する
b_linput	文字列を入力する
locate	カーソルを移動する
b_sprint	文字列をテキスト画面に表示する
b_iprint	数値をテキスト画面に表示する
b_fprint	数値をテキスト画面に表示する

3.16　BASICライブラリ

関数名	機　　能
b_tprint	テキスト画面上でタブスキップを行う
b_slprint	文字列をプリンタに出力する
b_ilprint	数値をプリンタに出力する
b_flprint	数値をプリンタに出力する
b_tlprint	タブスキップをプリンタに出力する
key	ファンクションキーに文字列をセットする
screen	表示画面のサイズ、色モード、解像度などを設定する
pi	円周率 π を返す
b_pi	円周率 π の倍数を返す
b_setdate	日付を設定する
b_settime	時刻を設定する
b_csw	カーソルを ON／OFF する
b_tpalet	テキストのパレット色を設定する
b_strncpy	文字列をコピーする
b_strcmp	文字列を比較する
b_stradd	2 つの文字列の連結を行う
b_init	BASIC の初期化を実行する
frename	ファイル名を変更する
fdelete	ファイルを削除する
apage	グラフィックのアクティブページを指定する
box	ボックスを設定する
circle	円弧を描く
contrast	テキスト画面のコントラストを設定する

3.16 BASICライブラリ

関数名	機　能
fill	ボックスを描きながらパレット色でその中を塗りつぶす
get	グラフィック画面の画像データを配列にとり込む
home	グラフィック画面の表示位置を設定する
hsv	色相、濃さ、明るさに応じた色コードを返す
line	線を引く
paint	指定された領域をパレット色で塗りつぶす
palet	パレットに色コードを割りあてる
point	グラフィック画面の指定座標のパレットコードを通知する
pset	グラフィック画面上に点を描く
put	グラフィック画面に画像データを表示する
rgb	赤、緑、青に応じた色コードを返す
symbol	グラフィック画面上に文字列を表示する
vpage	グラフィック画面の各ページの表示制御を行う
window	グラフィック画面のクリッピング領域の座標を設定する
wipe	グラフィック画面をクリアする
img_set	画像とり込みの設定を行う
img_still	画像とり込みの設定を行う
img_pos	画像とり込み時の画面水平位置を補正する
img_color	とり込み画像の色調補正を行う
v_cut	スーパーインポーズ時のテレビ画面の表示を制御する
crt	テレビ、ビデオ画面の切り替えを行う

関数名	機　能
img_ht	グラフィック画面とテレビ画面の半透明制御を行う
img_save	グラフィック画面のデータを指定ファイルに書き出す
img_load	グラフィック画面のデータを指定ファイルから読み込む
img_scrn	グラフィック画面のモード設定を行う
img_put	配列に格納されたデータをグラフィック画面に表示する
keysns	キー入力の有無を判定し、値を返す
img_home	グラフィック画面の表示位置を移動する
mouse	マウスの制御を行う
msarea	マウスカーソルの移動範囲を設定する
msbtn	マウスボタンが指定した状態になるまでの時間を返す
mspos	マウスカーソルの座標を変数に格納する
msstat	マウスの状態を変数に格納する
setmspos	マウスカーソルの位置を設定する
m_alloc	トラックバッファを確保し、それをクリアする
m_assign	FM 音源のチャンネルにトラックを割りあてる
m_cont	一時中断していた演奏を再開する
m_free	指定されたトラックの残り容量を通知する
m_init	FM 音源を初期化する
m_play	指定したチャンネルに割りあてたトラックデータを演奏する
m_stat	チャンネルの状態を通知する

関数名	機　能
m_stop	指定したチャンネルの演奏を一時中断する
m_tempo	テンポを指定する
m_trk	MML による楽譜データをトラックに書き込む
m_vget	音色データを配列に読み込む
m_vset	音色データを指定した音色番号のバッファに登録する
md_cont	一時中断していた演奏を再開する（MIDI 対応）
md_init	MIDI コントローラを初期化する（MIDI 専用）
md_off	MIDI チャンネルの出力の停止（MIDI 専用）
md_on	MIDI チャンネルの出力の開始（MIDI 専用）
md_play	指定したチャンネルに割りあてたトラックデータを演奏する（MIDI 対応）
md_regr	MIDI コントローラのレジスタからデータを読む（MIDI 専用）
md_regw	MIDI コントローラのレジスタにデータを書く（MIDI 専用）
md_stat	MIDI チャンネルの出力の禁止許可状態を得る（MIDI 専用）
md_stop	指定したチャンネルの演奏を一時中断する（MIDI 対応）
md_wrt	MIDI にデータを出力する（MIDI 専用）
sp_clr	スプライトをクリアする
sp_color	スプライトのパレット色を設定する
sp_def	スプライトのパターンデータを定義する
sp_disp	スプライト面の表示制御を行う
sp_move	スプライトプレーンの表示位置を設定する

3.16　BASICライブラリ

関数名	機　能
sp_off	スプライトの表示を OFF にする
sp_on	スプライトの表示を ON にする
sp_pat	スプライトパターンデータを配列に読み込む
sp_init	スプライトパターンを初期化する
sp_set	スプライトプレーンデータをセットする
sp_stat	指定されたスプライトの状態を通知する
bg_scroll	バックグラウンドのスクロール座標を指定する
bg_set	バックグラウンドの設定を行う
bg_stat	指定したバックグラウンドの状態を読み出す
bg_fill	バックグラウンドテキストをデータで埋めつくす
bg_put	バックグラウンドテキストに書き込む
bg_get	バックグラウンドテキストを読み出す
stick	ジョイスティックの状態を調べる
strig	指定したジョイスティックのボタンの状態を返す

3.17 IOCSコール

IOCSコールは、ROMに組み込まれたIOCSルーチンを呼び出すシステムコールです。
IOCSコールのおもな機能は以下の通りです。

- キーボード制御
- TVの表示制御
- テキストの表示制御
- RS-232C制御
- ジョイスティック制御
- プリンタ出力制御
- ディスクの入出力制御
- 日付、アラーム制御
- ADPCM制御
- FM音源制御
- 割り込み制御
- マウス制御
- DMA転送制御
- 日本語処理
- グラフィック制御
- スプライト制御

IOCSコールの関数には以下のものがあり、これらはすべてインクルードファイルiocslib. hで宣言されています。

関数名	機　　能
TRAP15	レジスタを直接指定しIOCSコールを行う
B_KEYINP	キーデータの読み出しを行う
B_KEYSNS	キーデータバッファ状態のセンスを行う
B_SFTSNS	シフトキーの押下状態を調べる
BITSNS	キーの押下状態を調べる

関数名	機　能
SKEYSET	指定キーを押したことにする
TVCTRL	TV コントロールコードを指定し、TV の操作を行う
LEDMOD	LED キーのモードを設定する
TGUSEMD	画面のモードを制御する
DEFCHR	指定の漢字パターンを外字エリアに書き込む
CRTMOD	CRT のモードを設定する
CONTRAST	コントラストを設定する
HSVTORGB	色相、濃さ、明るさに応じた色コードを作る
TPALET	テキストのパレットを指定する
TPALET2	テキストのパレットを指定する
TCOLOR	テキストのカラーを指定する
FNTGET	指定の漢字パターンを指定アドレスへ読み込む
TEXTGET	指定のドット座標からパターンを指定アドレスへ読み込む
TEXTPUT	指定アドレスからパターンを指定のドット座標へ書き込む
CLIPPUT	指定したクリップ座標に従いクリッピング処理を行う
SCROLL	テキスト／グラフィックの表示座標の設定を行う
B_CURON	カーソルを ON にする
B_CUROFF	カーソルを OFF にする
B_PUTC	指定した1バイトデータを表示する
B_PRINT	指定した文字列データを表示する
B_COLOR	カラー属性を設定する
B_LOCATE	カーソルを指定位置に設定する

3.17 IOCSコール

関数名	機　能
B_DOWN_S	1行下へカーソル移動する（最下行のときスクロール UP）
B_UP_S	1行上へカーソル移動する（最上行のときスクロール down）
B_UP	n行上へカーソル移動する
B_DOWN	n行下へカーソル移動する
B_RIGHT	n桁右へカーソル移動する
B_LEFT	n桁左へカーソル移動する
B_CLR_ED	カーソル位置から最終行右端まで画面をクリアする
B_CLR_ST	先頭行左端からカーソル位置まで画面をクリアする
B_CLR_AL	画面全体をクリアする
B_ERA_ED	カーソル位置から行の右端まで現在行をクリアする
B_ERA_ST	左端からカーソル位置まで現在行をクリアする
B_ERA_AL	現在行全体をクリアする
B_INS	現在カーソルが設定されている行にn行挿入する
B_DEL	現在カーソルが設定されている行からn行削除する
B_CONSOL	表示範囲を指定する
B_PUTMES	カーソル位置、カラー属性、表示文字数を指定して文字列を表示する
SET232C	RS-232C のモードを設定する
LOF232C	RS-232C の受信バッファ内のデータ数を求める
INP232C	RS-232C の受信データを読み込む

関数名	機　能
ISNS232C	RS-232C の受信データをチェックする
OSNS232C	RS-232C への送信が可能か否かチェックする
OUT232C	RS-232C への送信を行う
JOYGET	ジョイスティックのデータを読み込む
INIT_PRN	プリンタポートを初期化する
SNSPRN	プリンタ出力が可能か否かチェックする
OUTLPT	プリンタ出力を行う（データ直接）
OUTPRN	プリンタ出力を行う（シフト JIS で漢字処理あり）
B_SEEK	指定トラック（レコード）までシークする
B_VERIFY	データの比較チェックを行う
B_READDI	診断のためのチェックを行う（2HD のみ）
B_DSKINI	ディスクの初期化を行う
B_DRVSNS	ディスクのステータスを調べる
B_WRITE	ディスクにデータを書き込む
B_READ	ディスクからデータを読み込む
B_RECALI	トラック０へシークを行う
B_WRITED	削除データを書き込む（2HD のみ）
B_BADFMT	不良トラックを使用不可能にする
B_READDL	削除データを読み込む（2HD のみ）
B_FORMAT	物理フォーマットを行う
B_DRVCHK	指定ドライブの状態チェックと状態設定を行う
B_EJECT	イジェクトを行う
BINDATEBCD	日付を時計の形式に変換する

関数名	機　能
BINDATESET	日付を時計に設定する
TIMEBCD	時刻を時計の形式に変換する
TIMESET	時刻を時計に設定する
BINDATEGET	時計から日付を読み込む
DATEBIN	日付を BCD 表現から 2 進表現に変換する
TIMEGET	時計から時刻を読み込む
TIMEBIN	時刻を BCD 表現から 2 進表現に変換する
DATECNV	日付を表す文字列を 2 進データに変換する
TIMECNV	時刻を表す文字列を 2 進データに変換する
DATEASC	2 進データを、日付を表す文字列に変換する
TIMEASC	2 進データを、時刻を表す文字列に変換する
DAYASC	2 進データを、曜日を表す文字列に変換する
ALARMMOD	アラームの禁止／許可を設定する
ALARMSET	アラームの時刻と処理アドレスを設定する
ALARMGET	アラームの時刻と処理アドレスを読み込む
ADPCMOUT	ADPCM へデータを出力する
ADPCMINP	ADPCM からデータを入力する
ADPCMAOT	ADPCM へデータを出力する（アレイチェーンモード）
ADPCMAIN	ADPCM からデータを入力する（アレイチェーンモード）
ADPCMLOT	ADPCM へデータを出力する（リンクアレイチェーンモード）
ADPCMLIN	ADPCM からデータを入力する（リンクアレイチェーンモード）
ADPCMSNS	ADPCM の実行モードをセンスする

関数名	機　能
ADPCMMOD	ADPCMの実行を制御する
OPMSET	FM音源（YM2151）にデータを書き込む
OPMSNS	FM音源（YM2151）のステータスを読み込む
OPMINTST	FM音源ICによる割り込みを制御する
TIMERDST	MFPのTIMER-Dによる割り込みを制御する
VDISPST	V-DISPのカウンタの割り込みを制御する
CRTCRAS	CRTCに設定した割り込みラスタによる割り込みを制御する
HSYNCST	H-SYNCの起動による割り込みを制御する
PRNINTST	プリンタの割り込みを制御する
MS_INIT	マウスの初期化を行う
MS_CURON	マウスカーソルを表示する
MS_CUROF	マウスカーソルを消す
MS_STAT	マウスカーソルの表示モードを調べる
MS_GETDT	マウスの移動量とボタンのON／OFFを調べる
MS_CURGT	マウスカーソルの座標を調べる
MS_CURST	マウスカーソルの座標を設定する
MS_LIMIT	マウスカーソルの移動範囲を設定する
MS_OFFTM	マウスボタンを離すまでの時間を調べる
MS_ONTM	マウスボタンを押すまでの時間を調べる
MS_PATST	マウスカーソルのパターンを定義する
MS_SEL	マウスカーソルのパターンを選択する
MS_SEL2	マウスカーソルのパターンを複数指定してアニメーションを行う
SKEY_MOD	ソフトキーボードの制御を行う

関数名	機　能
DENSNS	電卓のセンスを行う
ONTIME	IOCSが起動してから現在までの時間を得る
B_INTVCS	ベクタの設定を行う
B_SUPER	スーパーバイザモードへの切り替えを行う
B_BPEEK	指定アドレスからデータを読み込む（バイト単位）
B_WPEEK	指定アドレスからデータを読み込む（ワード単位）
B_LPEEK	指定アドレスからデータを読み込む（ロングワード単位）
B_MEMSTR	指定アドレスからデータを読み込む（バイト数指定）
B_BPOKE	指定アドレスへデータを書き込む（バイト単位）
B_WPOKE	指定アドレスへデータを書き込む（ワード単位）
B_LPOKE	指定アドレスへデータを書き込む（ロングワード単位）
B_MEMSET	指定アドレスへデータを書き込む（バイト数指定）
DMAMOVE	DMA転送を行う
DMAMOV_A	DMA転送を行う（アレイチェーンによる）
DMAMOV_L	DMA転送を行う（リンクアレイチェーンによる）
DMAMODE	DMAの実行モードをセンスする
BOOTINF	ブート情報を得る
ROMVER	ROMのバージョンと作成年月日を得る

関数名	機　　能
G_CLR_ON	グラフィック画面をクリアして、表示モードにする
GPALET	グラフィックのパレットを設定する
SFTJIS	シフトJISコードをJIS漢字コードに変換する
JISSFT	JIS漢字コードをシフトJISコードに変換する
AKCONV	ANKコードをシフトJISコードに変換する
RMACNV	ローマ字変換を行う
DAKJOB	全角文字の濁点処理を行う
HANJOB	全角文字の半濁点処理を行う
OS_CURON	カーソルを表示する
OS_CUROF	カーソルを消す
APAGE	グラフィック画面の書き込みページを設定する
VPAGE	グラフィック画面の表示ページを設定する
HOME	グラフィック画面の表示位置を設定する
WINDOW	グラフィック画面にウィンドウを設定する
WIPE	グラフィック画面をクリアする
PSET	グラフィック画面に点を表示する
POINT	グラフィック画面上の点のパレットコードを得る
LINE	グラフィック画面にラインを引く
BOX	グラフィック画面にボックスを描く
FILL	グラフィック画面にボックスを描きその中を塗りつぶす
CIRCLE	グラフィック画面にサークルを描く
PAINT	グラフィック画面をペイントする
SYMBOL	グラフィック画面に文字を書く

3.17 IOCSコール

関数名	機　能
GETGRM	グラフィック画面の読み込みを行う
PUTGRM	グラフィック画面に書き込みを行う
SP_INIT	スプライト面の初期化を行う
SP_ON	スプライト面の表示を ON にする
SP_OFF	スプライト面の表示を OFF にする
SP_CGCLR	PCG をクリアする
SP_DEFCG	PCG を設定する
SP_GTPCG	PCG の読み出しを行う
SP_REGST	スプライトレジスタの設定を行う
SP_REGGT	スプライトレジスタの読み出しを行う
BGSCRLST	BG スクロールレジスタの設定を行う
BGSCRLGT	BG スクロールレジスタの読み出しを行う
BGCTRLST	BG コントロールレジスタの設定を行う
BGCTRLGT	BG コントロールレジスタの読み出しを行う
BGTEXTCL	BG テキストを指定のコードで埋める
BGTEXTST	BG テキストを設定する
BGTEXTGT	BG テキストの読み出しを行う
SPALET	スプライトパレットの設定と読み出しを行う
TXXLINE	テキスト画面に X 方向ラインを引く
TXYLINE	テキスト画面に Y 方向ラインを引く
TXBOX	テキスト画面にボックスを描く
TXFILL	テキスト画面にボックスを描きその中を塗りつぶす
TXREV	テキスト画面の反転を行う

関数名	機　　能
TXRASCPY	テキスト画面でラスタコピーを行う
ABORTRST	アボートするためのフラグをリセットする（OSで使用）
IPLERR	IPL時のエラー処理を行う
ABORTJOB	アボート処理を行う（OSで使用）

3.18 DOSコール

DOS コールは、Human68k のサブルーチンを呼び出すシステムコールのことであり、おもな機能は以下の通りです。

- RS-232C 回線の入出力制御
- キーボード入力制御
- 漢字変換行の制御
- コンソール直接入出力
- 日付、時刻の制御
- ファイルの入出力制御
- デバイスドライバの直接入出力
- プログラムのロードと実行
- 環境変数の制御

DOS コールの関数には以下のものがあり、これらはすべてインクルードファイル doslib. h で宣言されています。

関数名	機　能
EXIT	プログラムを終了する
GETCHAR	キーボードより文字コードを入力し、表示する
PUTCHAR	文字コードを表示する
COMINP	AUX（RS-232C）より1文字入力する
COMOUT	AUX（RS-232C）へ1文字出力する
PRNOUT	プリンタへ1文字出力する
INPOUT	キー入力／センスまたは文字コードの表示を行う
INKEY	文字コードを入力する（ブレークチェックなし）
GETC	文字コードを入力する（ブレークチェックあり）
PRINT	文字列を表示する
GETS	文字列を入力する（ブレークチェックあり）

3.18　DOSコール

関数名	機　　能
KEYSNS	キーの入力状態を調べる
KFLUSHGP	先行入力を捨ててキーボード入力を行う
KFLUSHIO	先行入力を捨ててキーボード入力を行う
KFLUSHIN	先行入力を捨ててキーボード入力を行う
KFLUSHGC	先行入力を捨ててキーボード入力を行う
KFLUSHGS	先行入力を捨ててキーボード入力を行う
FFLUSH	ディスクをリセットする
CHGDRV	カレントドライブを指定する
DRVCTRL	指定ドライブの状態のチェックと設定を行う
CONSNS	画面への出力が可能であるか否か調べる
PRNSNS	プリンタへの出力が可能であるか否か調べる
CINSNS	AUX（RS-232C）からの入力が可能であるか否か調べる
COUTSNS	AUX（RS-232C）への出力が可能であるか否か調べる
FATCHK	指定ファイルのドライブ番号とセクタの関連を調べる
FATCHK2	大容量の指定ファイルのドライブ番号とセクタの関連を調べる
HENDSPMO	漢字変換行の制御（モード表示ウィンドウのオープン）
HENDSPMP	漢字変換行の制御（モード表示のウィンドウへのノーマル表示）
HENDSPMR	漢字変換行の制御（モード表示ウィンドウへのリバース表示）
HENDSPMC	漢字変換行の制御（モード表示ウィンドウのクローズ）

関数名	機　能
HENDSPIO	漢字変換行の制御（入力ウィンドウのオープン）
HENDSPIP	漢字変換行の制御（入力ウィンドウのノーマル表示）
HENDSPIR	漢字変換行の制御（入力ウィンドウのリバース表示）
HENDSPIC	漢字変換行の制御（入力ウィンドウのクローズ）
HENDSPSO	漢字変換行の制御（候補ウィンドウのオープン）
HENDSPSP	漢字変換行の制御（候補ウィンドウへのノーマル表示）
HENDSPSR	漢字変換行の制御（候補ウィンドウへのリバース表示）
HENDSPSC	漢字変換行の制御（候補ウィンドウのクローズ）
CURDRV	カレントドライブを得る
GETSS	文字列を入力する（ブレークチェックなし）
FGETC	指定したファイルハンドルから1文字入力を行う
FGETS	指定したファイルハンドルから文字列を入力する
FPUTC	指定した文字コードをファイルハンドルへ出力する
FPUTS	指定した文字列をファイルハンドルへ出力する
ALLCLOSE	すべてのファイルハンドルをクローズする
SUPER	スーパーバイザモード／ユーザーモードの切り替えを行う
FNCKEYGT	再定義可能キーの読み出しを行う
FNCKEYST	再定義可能キーの設定を行う
C_PUTC	コンソール直接出力（指定した1バイトデータの表示）

関数名	機　能
C_PRINT	コンソール直接出力（指定した文字列データの表示）
C_COLOR	コンソール直接出力（指定した属性の設定）
C_LOCATE	コンソール直接出力（カーソル位置の設定）
C_DOWN_S	コンソール直接出力（1行下へカーソル移動）(最下行のときスクロールUP)
C_UP_S	コンソール直接出力（1行上へカーソル移動）(最上行のときスクロールDOWN)
C_UP	コンソール直接出力（n行上へカーソル移動）
C_DOWN	コンソール直接出力（n行下へカーソル移動）
C_RIGHT	コンソール直接出力（n桁右へカーソル移動）
C_LEFT	コンソール直接出力（n桁左へカーソル移動）
C_CLS_ED	コンソール直接出力（カーソル位置から最終行右端までクリア）
C_CLS_ST	コンソール直接出力（先頭行左端からカーソル位置までクリア）
C_CLS_AL	コンソール直接出力（画面全体をクリア）
C_ERA_ED	コンソール直接出力（カーソル位置から行の右端までクリア）
C_ERA_ST	コンソール直接出力（行の左端からカーソル位置までクリア）
C_ERA_AL	コンソール直接出力（カーソルのある行全体をクリア）
C_INS	コンソール直接出力（カーソルのある行にn行挿入）
C_DEL	コンソール直接出力（カーソルのある行からn行削除）

関数名	機　　能
C_FNKMOD	コンソール直接出力（ファンクションキー行のモード設定）
C_WINDOW	コンソール直接出力（スクロール範囲の設定）
C_WIDTH	コンソール直接出力（画面サイズの設定）
C_CURON	コンソール直接出力（カーソル表示モードの設定）
C_CUROFF	コンソール直接出力（カーソル非表示モードの設定）
K_KEYINP	コンソール直接入力（キーボード入力）
K_KEYSNS	コンソール直接入力（キーコードの先読み）
K_SFTSNS	コンソール直接入力（シフトキー状態のセンス）
K_KEYBIT	コンソール直接入力（キーコードグループの状態通知）
K_INSMOD	コンソール直接入力（挿入モードのON／OFF設定）
INTVCS	ベクタにアドレスを設定する
PSPSET	プロセス管理情報を書き込む
GETTIM2	現在の時刻を求める
SETTIM2	時刻を設定する
NAMESTS	ファイル名に関する情報を得る
GETDATE	現在の日付を求める
SETDATE	日付を設定する
GETTIME	現在の時刻を求める
SETTIME	時刻を設定する

関数名	機　能
VERIFY	ベリファイフラグの設定を行う
DUP0	ファイルハンドルの強制複写
VERNUM	Human68k のバージョン番号を得る
KEEPPR	プログラムを常駐終了する
GETDPB	指定装置番号のドライブパラメータブロックをコピーする
BREAKCK	ブレークチェックの設定を行う
DRVXCHG	ドライブ名の入れ替えを行う
INTVCG	ベクタの値を求める
DSKFRE	ディスクの残り容量を求める
NAMECK	ファイル名をバッファに展開する
MKDIR	ディレクトリを作成する
RMDIR	ディレクトリを削除する
CHDIR	カレントディレクトリを変更する
CREATE	ファイルを作成する
OPEN	ファイルをオープンする
CLOSE	ファイルをクローズする
READ	ファイルから読み込みを行う
WRITE	ファイルへ書き出しを行う
DELETE	ファイルを削除する
SEEK	ファイルポインタを移動する
CHMOD	ファイルの属性を変更する
IOCTRLGT	デバイスドライバの直接入出力（装置情報の通知）

3.18 DOSコール

関数名	機　能
IOCTRLST	デバイスドライバの直接入出力（装置情報の設定）
IOCTRLRH	デバイスドライバの直接入出力（データの読み込み）
IOCTRLWH	デバイスドライバの直接入出力（データの書き出し）
IOCTRLRD	デバイスドライバの直接入出力（装置番号からの読み込み）
IOCTRLWD	デバイスドライバの直接入出力（装置番号からの書き出し）
IOCTRLIS	デバイスドライバの直接入出力（入力ステータスの通知）
IOCTRLOS	デバイスドライバの直接入出力（出力ステータスの通知）
IOCTRLDVGT	指定ドライブがローカルかリモートかを調べる
IOCTRLFDGT	指定ファイルがローカルかリモートかを調べる
IOCTRLRTSET	リトライ数を設定する
IOCTRLDVCTL	指定ドライブの特殊コントロールを行う
IOCTRLFDCTL	指定ファイルの特殊コントロールを行う
DUP	ファイルハンドルを複写する
DUP2	ファイルハンドルの強制複写を行う
CURDIR	カレントディレクトリを得る
MALLOC	メモリ領域を確保する
MFREE	メモリ領域を解放する
SETBLOCK	メモリブロックのサイズを変更する

3.18　DOSコール

関数名	機　　能
EXEC2	プログラムのロード、実行、モジュール番号の取得を行う
LOADEXEC	プログラムをロードし実行する
LOAD	プログラムをロードする
PATHCHK	プログラムロードのための準備作業を行う
LOADONLY	プログラムをロードする
EXECONLY	ロード後のプログラムを実行する
BINDNO	プログラムのモジュール番号の取得を行う
EXIT2	プログラムを終了する
WAIT	プロセスの終了コードを得る
FILES	属性に従ってファイルを検索する
NFILES	属性に従って次のファイルを検索する
SETPDB	指定したプロセスに制御をわたす
GETPDB	現在のプロセスのアドレスを求める
SETENV	指定された環境に変数を設定する
GETENV	指定された環境から指定された変数の内容をとり出す
VERIFYG	ベリファイフラグの設定状況を調べる
COMMON_CK	コモン領域が存在するかどうか調べる
COMMON_RD	コモン領域からデータの読み込みを行う
COMMON_WT	コモン領域の作成およびデータの書き込みを行う
COMMON_LK	コモン領域をロックする
COMMON_FRE	コモン領域のロックを削除する
COMMON_DEL	コモン領域を削除する
MOVE	ファイルを移動する

3.18 DOSコール

関数名	機　能
RENAME	ファイル名を変更する
FILEDATE	ファイルの作成日付／時刻の読み出しと設定
MALLOC2	割り当てのモードを指定してメモリを割り当てる
MAKETMP	作業用ファイルを作成する
NEWFILE	新規にファイルを作成する
LOCK	ファイルの共有を禁止する
UNLOCK	ファイルの共有の禁止を解除する
GETASSIGN	仮想割り当ての状態を得る
MAKEASSIGN	仮想割り当てを作成する
RASSIGN	仮想割り当てを解除する
S_MALLOC	主スレッドのメモリからメモリを割り当てる
S_MFREE	主スレッドのメモリから割り当てたメモリを開放する
S_PROCESS	サブスレッドのメモリ領域を設定する
DISKRED	ブロックデバイスの直接リード
DISKRED2	大容量ブロックデバイスの直接リード
DISKWRT	ブロックデバイスの直接ライト
DISKWRT2	大容量ブロックデバイスの直接ライト
INDOSFLG	OS の内部ワークのアドレスを得る
SUPER_JSR	スーパーバイザ領域のサブルーチンを呼び出す
BUS_ERR	バスエラーが発生するかを調べ、メモリ転送を行う
OPEN_PR	サブスレッドを作成する
KILL_PR	サブスレッドを削除する
GET_PR	スレッド情報を得る

3.18　DOSコール

関数名	機　　能
SUSPEND_PR	スレッドを強制的にスリープ状態にする
SLEEP_PR	スレッドをスリープ状態にする
SEND_PR	スレッド間のコマンド、データ転送を行う
TIME_PR	スレッド管理用のタイマーの値を得る
CHANGE_PR	スレッドの実行権を放棄する

3.19 その他

これらの関数は、他のカテゴリーには分類できないものです。機能のカッコ内は、宣言されているインクルードファイル名です。

関数名	機　能
abs	int 型整数値の絶対値を求める (stdlib. h)
labs	long 型整数値の絶対値を求める (stdlib. h)
wabs	short 型整数値の絶対値を求める (stdlib. h)
assert	論理エラーをテストする (assert. h)
getenv	環境変数の値を得る (stdlib. h)
longjmp	セーブされたスタック環境をリストアする (setjmp. h)
max	2つの数の大きい方を求める (stdlib. h)
mix	2つの数の小さい方を求める (stdlib. h)
offsetof	構造体のメンバの位置を得る (stddef. h)
perror	エラーメッセージを表示する (stdio. h)
strerror	エラーメッセージを得る (string. h)
putenv	環境変数の値を修正／追加する (stdlib. h)
srand	疑似乱数を初期化する (stdlib. h)
rand	擬似乱数を得る (stdlib. h)
setjmp	スタック環境をセーブする (setjmp. h)
va_arg	可変引数の内容を取り出す (stdarg. h)
va_end	可変引数の取り出しを終わる (stdarg. h)
va_start	可変引数の取り出しを始める (stdarg. h)

第4章

リファレンス

この章では、XC コンパイラがサポートするライブラリ関数（小文字）を、リファレンス形式でアルファベット順に掲載しています。
関数には、レベルが設定されています。
これは、C 言語で一般的にいわれている入出力のレベルとは違い、XC コンパイラ独自のものです。
レベルが表す意味は、次の通りです。

レベル 3：BASIC 関連の関数（ライブラリファイル BASLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。
レベル 2：ライブラリファイル CLIB. L に含まれるすべての関数、および、これらの関数と混在させて使用できるもの。
レベル 1：DOS コール関連の関数（ライブラリファイル DOSLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。
レベル 0：IOCS コール関連の関数（ライブラリファイル IOCSLIB. L に含まれる）のうち、他のレベルの関数と混在させて使うときには、注意しなければならないもの。

レベル 2 以外の関数を使用するときは、なるべく同一レベルの関数だけでプログラムを記述してください。

abort

レベル2

書　式

#include <stdlib. h>

void abort( ) ;

機　能

現在実行中のプロセスを終了し、呼び出し元のプロセス（通常オペレーティングシステム）に戻ります。
このとき、呼び出し元のプロセスには終了ステータス3が返されます。
また、次のメッセージを標準エラー出力（stderr）に出力します。

Abnormal program termination

なお、abortを呼び出してもストリームバッファの内容はフラッシュされません。

戻り値

戻り値はありません。

参照関数

exit, _exit

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        char    *name;
        FILE    *fp;

        name = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(name, "r")) == NULL) {
                perror(name);
                abort();
        }
        printf("open secceeded¥n");
        (void)fclose(fp);
}
```

abs, labs, wabs　レベル2

書　　式　# include 〈stdlib. h〉

int abs(n) ; /* int 型の整数値の絶対値を求める */
int n; /* int 型の整数値 */

long labs(n) ; /* long 型の整数値の絶対値を求める */
long n; /* long 型の整数値 */

short wabs(n) ; /* short 型の整数値の絶対値を求める */
short n; /* short 型の整数値 */

機　　能　整数 n の絶対値を求めます。

戻 り 値　整数 n の絶対値を返します。

参照関数　cabs, fabs

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        int     i;
        long    l;

        for (i = -10; i <= 10; i++)
                printf("abs(%d) = %d¥n", i, abs(i));
        for (l = -10; l <= 10L; l++)
                printf("labs(%ld) = %ld¥n", l, labs(l));
}
```

access

レベル2

書　式

```
#include <io.h>

int access(pathname, mode);
char *pathname;  /* ファイルのパス名 */
int mode;  /* ファイルのアクセスモード */
```

機　能

pathnameで指定されたファイルがmodeで指定したモードでアクセスできるかどうかを調べます。
modeの値とその内容を以下に示します。

modeの値	内　容
06	読み込み、書き込みともに可能かどうかを調べる
04	読み込み可能かどうかを調べる
02	書き込み可能かどうかを調べる
00	ファイルが存在するかどうかを調べる

Human68kでは、ファイルとして存在すれば必ず読むことができますので、00と04は同じ結果になります。
同様に02と06も同じ結果になります。

戻り値

ファイルが指定したアクセスモードをもつときには0を返します。
ファイルが存在しないとき、または指定したモードではアクセスできないときには−1が返され、同時にerrnoに次の値のうちのいずれかがセットされます。

errnoの値	内　容
EACCES	modeで指定したアクセスはできない
ENOENT	ファイル名またはパス名がみつからない

参照関数

chmod, fstat, open, stat

プログラム例

```
#include        <stdio.h>
#include        <io.h>


main()
{
        char    *filename = "file";

        if ((access(filename, 0)) != 0)
                printf("%s: file が存在しない。¥n", filename);
        else {
                if ((access(filename, 4)) == 0)
                        printf("%s: 読み込み可能¥n", filename);
                else
                        printf("%s: 読み込み不可¥n", filename);

                if ((access(filename, 2)) == 0)
                        printf("%s: 書き込み可能¥n", filename);
                else
                        printf("%s: 書き込み不可¥n", filename);
        }

}
```

a_cont

レベル3

書　式　#include <audio. h>

void a_cont( ) ;

機　能　中断されていた PCM の動作を再開します。

戻り値　戻り値はありません。

参照関数　a_end, a_play, a_rec, a_stat, a_stop

プログラム例

```
#include        <stdio.h>
#include        <audio.h>

#define KHz3_9          0
#define KHz5_2          1
#define KHz7_8          2
#define KHz10_4         3
#define KHz15_6         4
#define M_CUT           0
#define M_LEFT          1
#define M_RIGHT         2
#define M_STEREO        3
#define SIZE            32732
#define A_SIZE          65536
#define NASI            'NASI'
#define ST_NON          0        /* 何も実行していない */
#define ST_OUT          2        /* 出力中(ADPCMOUT, a_play) */
#define ST_INP          4        /* 入力中(ADPCMINP, a_rec) */
#define ST_AOT          12       /* 出力中(ADPCMAOT) */
#define ST_AIN          14       /* 入力中(ADPCMAIN) */
#define ST_LOT          22       /* 出力中(ADPCMLOT) */
#define ST_LIN          24       /* 入力中(ADPCMLIN) */

static char     aca[A_SIZE];


main()
{

        a_rec(aca, A_SIZE, KHz15_6, NASI);
        while (a_stat() != ST_NON)
                puts("recording...");

        a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
        if (a_stat() != ST_NON) {
                puts("playing...");
                a_stop();
                puts("paused...");
                a_cont();
                puts("contune...");
        }

}
```

acos

レベル2

書　式	#include <math. h> double acos(x) ; double x ;
機　能	x (−1≦x≦1) のアークコサインを計算します。
戻り値	xのアークコサインの値 (0〜π) を返します。 xが−1より小さいか1より大きい場合は0を返し、errnoにEDOMがセットされます。 エラー処理はmatherrルーチンで行うことができます。
参照関数	asin, atan, atan2, cos, matherr, sin, tan

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        int     i;

        for (i = -10; i <= 10; i++) {
                x = (double)i / 10.0;
                printf("acos(%+4.1f) = %g¥n", x, acos(x));
        }
}
```

a_end

レベル3

書　式　# include <audio. h>

void a_end( ) ;

機　能　PCM の動作を強制終了します。

戻り値　戻り値はありません。

参照関数　a_cono, a_play, a_rec, a_stat, a_stop

プログラム例

```
#include        <stdio.h>
#include        <audio.h>

#define KHz3_9          0
#define KHz5_2          1
#define KHz7_8          2
#define KHz10_4         3
#define KHz15_6         4
#define M_CUT           0
#define M_LEFT          1
#define M_RIGHT         2
#define M_STEREO        3
#define SIZE            32732
#define A_SIZE          65536
#define NASI            'NASI'
#define ST_NON          0        /* 何も実行していない */
#define ST_OUT          2        /* 出力中(ADPCMOUT, a_play) */
#define ST_INP          4        /* 入力中(ADPCMINP, a_rec) */
#define ST_AOT          12       /* 出力中(ADPCMAOT) */
#define ST_AIN          14       /* 入力中(ADPCMAIN) */
#define ST_LOT          22       /* 出力中(ADPCMLOT) */
#define ST_LIN          24       /* 入力中(ADPCMLIN) */

static char     aca[A_SIZE];


main()
{

        a_rec(aca, A_SIZE, KHz15_6, NASI);
        while (a_stat() != ST_NON)
                puts("recording...");

        a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
        if (a_stat() != ST_NON) {
                puts("playing...");
                a_end();
                puts("aborted...");
        }

}
```

apage　　レベル3

書　　式　#include <graph. h>

void apage(page) ;
char page ; /* アクティブページ */

機　　能　グラフィック画面のアクティブページを設定します。
なお、ページ数は画面のモードによって異なります。

戻 り 値　戻り値はありません。

参照関数　screen, vpage

プログラム例

```
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i;

        b_init();

        screen(1, 1, 1, 1);
        vpage(0);

        apage(0);
        fill(50, 50, 200, 200, 3);
        apage(1);
        fill(100, 100, 250, 250, 5);
        apage(2);
        fill(150, 150, 300, 300, 9);
        apage(3);
        fill(200, 200, 350, 350, 15);

        for (i = 0; i < 10000; i++)
                vpage(0x01);
        for (i = 0; i < 10000; i++)
                vpage(0x02);
        for (i = 0; i < 10000; i++)
                vpage(0x04);
        for (i = 0; i < 10000; i++)
                vpage(0x08);
        vpage(0x0f);

        b_exit(0);
}
```

a_play

レベル3

書式

```
#include <audio.h>

void a_play(buf, size, cl, md, length) ;
void *buf ;  /*  PCMデータ格納配列へのポインタ  */
int size ;  /*  bufのバイトサイズ  */
int cl ;  /*  サンプリング周波数  */
int md ;  /*  出力モード  */
int length ;  /*  有効バイト数  */
```

機能

PCM録音の再生を行います。
bufにはPCMデータ格納配列のアドレスを指定します。
sizeにはbufのバイト数を指定します。
clにはサンプリング周波数を指定します。

0　:　3.9KHz
1　:　5.2KHz
2　:　7.8KHz
3　:　10.4KHz
4　:　15.6KHz

mdには出力モードを指定します。

0　:　カット
1　:　左
2　:　右
3　:　ステレオ(左右両方)

lengthには有効バイト数を指定します。指定したところまでが再生されます。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <conio.h>
#include        <audio.h>

#define KHz3_9          0
#define KHz5_2          1
#define KHz7_8          2
#define KHz10_4         3
#define KHz15_6         4
#define M_CUT           0
#define M_LEFT          1
#define M_RIGHT         2
#define M_STEREO        3
#define SIZE            32732
#define A_SIZE          65536
#define NASI            'NASI'

static char     aca[A_SIZE];


main()
{
        hit_any_key("PCM record?");
        a_rec(aca, A_SIZE, KHz15_6, NASI);

        hit_any_key("PCM play?");
        a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
}


hit_any_key(char *str)
{
        printf("%s  hit any key¥n", str);
        (void)getch();
}
```

asc

レベル3

書式

```
#include <basic.h>

int asc(s);
char *s;  /* 文字列 */
```

機能 文字列sの先頭1バイトの文字コードを返します。

戻り値 文字列sの先頭1バイトの文字コードを返します。

a_rec　レベル 3

書　　式

```
#include <audio.h>

void a_rec(buf, size, cl, length);
void *buf;  /* PCMデータ格納配列へのポインタ */
int size;  /* bufのバイトサイズ */
int cl;  /* サンプリング周波数 */
int length;  /* 有効バイト数 */
```

機　　能

PCM録音を行い、そのデータを配列に格納します。
bufにはPCMデータを格納する配列のアドレスを指定します。
sizeにはbufのバイト数を指定します。
clにはサンプリング周波数を指定します。

0 : 3.9KHz
1 : 5.2KHz
2 : 7.8KHz
3 : 10.4KHz
4 : 15.6KHz

lengthには有効バイト数を指定します。
指定したところまで録音します。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <conio.h>
#include        <audio.h>

#define KHz3_9          0
#define KHz5_2          1
#define KHz7_8          2
#define KHz10_4         3
#define KHz15_6         4
#define M_CUT           0
#define M_LEFT          1
#define M_RIGHT         2
#define M_STEREO        3
#define SIZE            32732
#define A_SIZE          65536
#define NASI            'NASI'

static char     aca[A_SIZE];


main()
{
        hit_any_key("PCM record?");
        a_rec(aca, A_SIZE, KHz15_6, NASI);

        hit_any_key("PCM play?");
        a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
}


hit_any_key(char *str)
{
        printf("%s  hit any key¥n", str);
        (void)getch();
}
```

asctime

レベル2

書式

#include <time. h>

char ＊asctime(time) ;
struct tm ＊time ; /＊ tm構造体を指すポインタ ＊/

機能

tm構造体に格納されている時間情報を、文字列に変換します。
得られる文字列は26文字で、フォーマットは以下のようになります。

wed Oct 22 13 : 15 : 30 1986¥n¥0

なお、時刻は24時間表示です。

asctimeは、内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。
また、前回の呼び出しで返された文字列は失われます。

戻り値

変換された文字列へのポインタを返します。

参照関数

ctime, ftime, gmtime, time, tzset

プログラム例

```
#include        <stdio.h>
#include        <time.h>


main()
{
        struct tm       *ntime;
        time_t  ltime;

        time(&ltime);
        ntime = localtime(&ltime);
        printf("ただ今の時刻は %s", asctime(ntime));
}
```

asin

レベル2

書　式

#include <math. h>

double asin(x) ;
double x ;

機　能

x (−1≦x≦1) のアークサインを計算します。

戻り値

xのアークサインの値 (−π/2〜π/2) を返します。
xが−1より小さいか1より大きい場合は0を返し、errnoにEDOMがセットされます。
エラー処理はmatherr関数を使って行うことができます。

参照関数

acos, atan, atan2, cos, matherr, sin, tan

プログラム例

```
#include        <stdio.h>
#include        <math.h>

main()
{
        double  x;
        int     i;

        for (i = -10; i <= 10; i++) {
                x = (double)i / 10.0;
                printf("asin(%+4.1f) = %g¥n", x, asin(x));
        }
}
```

assert

レベル2

書式

\#include <assert. h>
\#include <stdio. h>

void assert(expression) ; /* expression＝論理式 */

機能

assert マクロは、expression が偽 (0) の場合に、診断メッセージを出力し、プロセスを終了します。
診断メッセージは下記の形式です。

Assertion failed: expression, file filename, line linenumber

expression は論理式、filename はソースファイル名で、linenumber は expression が偽になった行番号です。
expression が真 (0 以外) のときは、何も行いません。
通常 assert は、デバッグ時にプログラムの論理エラーの判定のために使用します。
デバッグの終了後、NDEBUG を使って assert をプログラムから取り除きます。NDEBUG をコマンド行のオプション/D で指定するか、#define プリプロセッサ命令を使って定義すると、コンパイル時にプリプロセッサによりすべての assert 呼び出しがソースプログラムから取り除かれます。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <assert.h>


main()
{
        check_string("str", 3);
        check_string(NULL, 0);
        check_string("", 3);
        check_string("str", 0);
}


check_string(char *str, int len)
{
        assert(str != NULL);
        assert(*str != '¥0');
        assert(len > 0);
}
```

a_stat

レベル3

書　式

```
#include <audio.h>

int a_stat( ) ;
```

機　能

現在の PCM の動作状況を調べ、戻り値として返します。

戻り値

PCM の実行状態を以下に示す値で返します。

0x00 : 何も実行していない
0x02 : 出力中(ADPCMOUT, a_play を実行中)
0x04 : 入力中(ADPCMINP, a_rec を実行中)
0x12 : 出力中(ADPCMAOT を実行中)
0x14 : 入力中(ADPCMAIN を実行中)
0x22 : 出力中(ADPCMLOT を実行中)
0x24 : 入力中(ADPCMLIN を実行中)

なお、a_stat の戻り値は、ADPCMSNS 関数の戻り値と同じです。

参照関数

a_cont, a_end, a_play, a_rec, a_stop, ADPCMSNS

プログラム例

```
#include        <staio.h>
#include        <audio.h>

#define KHz3_9          0
#define KHz5_2          1
#define KHz7_8          2
#define KHz10_4         3
#define KHz15_6         4
#define M_CUT           0
#define M_LEFT          1
#define M_RIGHT         2
#define M_STEREO        3
#define SIZE            32732
#define A_SIZE          65536
#define NASI            'NASI'
#define ST_NON          0       /* 何も実行していない */
#define ST_OUT          2       /* 出力中(ADPCMOUT, a_play) */
#define ST_INP          4       /* 入力中(ADPCMINP, a_rec) */
#define ST_AOT          12      /* 出力中(ADPCMAOT) */
#define ST_AIN          14      /* 入力中(ADPCMAIN) */
#define ST_LOT          22      /* 出力中(ADPCMLOT) */
#define ST_LIN          24      /* 入力中(ADPCMLIN) */

static char     aca[A_SIZE];


main()
{
        a_rec(aca, A_SIZE, KHz15_6, NASI);
        while (a_stat() != ST_NON)
                puts("recording...");

        a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
        while (a_stat() != ST_NON)
                puts("playing...");
}
```

a_stop

レベル 3

書　　式 #include <audio. h>

void a_stop( ) ;

機　　能 PCM の動作を中断します。
a_cont 関数で再開することができます。

戻 り 値 戻り値はありません。

参照関数 a_cont, a_end, a_play, a_rec, a_stat

プログラム例

```
#include        <stdio.h>
#include        <audio.h>

#define KHz3_9          0
#define KHz5_2          1
#define KHz7_8          2
#define KHz10_4         3
#define KHz15_6         4
#define M_CUT           0
#define M_LEFT          1
#define M_RIGHT         2
#define M_STEREO        3
#define SIZE            32732
#define A_SIZE          65536
#define NASI            'NASI'
#define ST_NON          0       /* 何も実行していない */
#define ST_OUT          2       /* 出力中(ADPCMOUT, a_play) */
#define ST_INP          4       /* 入力中(ADPCMINP, a_rec) */
#define ST_AOT          12      /* 出力中(ADPCMAOT) */
#define ST_AIN          14      /* 入力中(ADPCMAIN) */
#define ST_LOT          22      /* 出力中(ADPCMLOT) */
#define ST_LIN          24      /* 入力中(ADPCMLIN) */

static char     aca[A_SIZE];


main()
{
        a_rec(aca, A_SIZE, KHz15_6, NASI);
        while (a_stat() != ST_NON)
                puts("recording...");

        a_play(aca, A_SIZE, KHz15_6, M_STEREO, SIZE);
        if (a_stat() != ST_NON) {
                puts("playing...");
                a_stop();
                puts("paused...");
                a_cont();
                puts("contune...");
        }
}
```

atan, atan2

レベル2

書　式

```
#include <math.h>

double atan(x) ;
double x ;

double atan2(x, y) ;
double x ;
double y ;
```

機　能

atan は x のアークタンジェントを計算します。
atan2 は x/y のアークタンジェントを計算します。

戻り値

atan は x のアークタンジェントの値（$-\pi/2 \sim \pi/2$）を、atan2 は x/y のアークタンジェントの値（$-\pi \sim \pi$）をそれぞれ返します。
引数が許される範囲にない場合は、返り値は不定となり、errno に EDOM がセットされます。
エラー処理は matherr 関数を使って行うことができます。

参照関数

acos, asin, cos, matherr, sin, tan

プログラム例

```
#include        <stdio.h>
#include        <math.h>

main()
{
        double  x;
        double  y;
        int     i;
        int     j;

        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("atan(%+4.1f) = %g¥n", x, atan(x));
        }
        for (i = -32; i <= 32; i++) {
                for (j = -32; j <= 32; j++) {
                        x = (double)i / 10.0;
                        y = (double)j / 10.0;
                        printf("atan2(%+4.1f, %+4.1f) = %g¥n",
                         y, x, atan2(y, x));
                }
        }
}
```

atexit

レベル2

書　式　#include <stdlib. h>

int atexit(void(＊ func)( )) ;
void (＊ func)( ) ;　/＊　終了時に呼び出される関数へのポインタ　＊/

機　能　funcで指定した関数を、プログラム終了時に呼び出す関数として登録します。
exitによってプログラムが正常終了すると、呼び出し元の親プロセスに戻る前に(＊ func)( )を呼び出します。
呼び出される関数は、引数なしで、戻り値を持たない関数です。
atexitが呼び出されるたびに、新たな関数を追加・登録します。
複数の関数が登録された場合、終了時に実行される順序は登録された順番と逆の順番です。
登録できる関数の個数は最大32個までです。

戻り値　成功すると0を、失敗すると(32個を超えた場合など)0以外を返します。

参照関数　exit

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


void    func1(void)
{
        printf("終了時の処理その1¥n");
}


void    func2(void)
{
        printf("終了時の処理その2¥n");
}


main()
{
        if (atexit(func1) != 0)
                printf("登録できません。¥n");
        if (atexit(func2) != 0)
                printf("登録できません。¥n");
        exit(0);
}
```

atof, atoi, atol, atow

レベル２

書　式

```
#include <stdlib.h>

double atof(string);
int atoi(string);
long atol(string);
short atow(string);
char *string;  /* 変換される文字列へのポインタ */
```

機　能

atof は文字列 string を倍精度（double 型）の浮動小数点値に変換します。
atoi は文字列 string を整数値に変換します。
atol は文字列 string を long 型整数値に変換します。
atow は文字列 string を short 型整数値に変換します。
string は数値に変換できる一連の文字列で、数値または数値の一部として解釈できない最初の文字までが変換の対象になります。
変換できる文字列の形式を以下に示します。

【atof】

［空白（またはタブ文字）］［符号］［数字］［. 数字］［E［符号］数字］
［空白（またはタブ文字）］は無視されます。
［符号］は"＋"または"－"です。"＋"のときは省略できます。
［数字］は 1 桁以上の 10 進数です。これがないときは、［. 数字］に少なくとも 1 桁の数字が必要です。
［. 数字］は小数点以下の数字です。
また、－123E－12 のように指数表現することもできます。

【atoi, atol, atow】

［空白（またはタブ文字）］［符号］数字
各々の意味は atof と同様です。

戻り値

それぞれ変換した数値を返します。
変換できなかった場合は 0 を返します。
またオーバーフローした場合の戻り値は不定です。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        printf("%g¥n", atof("1.49597870e+11"));
        printf("%d¥n", atoi("09471"));
        printf("%ld¥n", atol("037971030"));
        printf("%d¥n",atow("10287"));
}
```

b_binS

レベル 3

書　式

```
#include <basic.h>

char *b_binS(s, n);
char *s;  /* 文字列格納領域へのポインタ */
int n;  /* 数値 */
```

機　能

数値nを2進数表現の文字列に変換し、sが指す領域へ格納します。
sが指す領域は、最低34バイト以上なければなりません。

戻り値

変換結果の文字列へのポインタを返します。

b_chrS

レベル3

書式

```
#include <basic.h>

char *b_chrS(s, ch) ;
char *s; /* 文字列格納領域へのポインタ */
int ch; /*  文字コード */
```

機能 文字コードを文字列に変換します。

戻り値 変換結果の文字列へのポインタを返します。

b_csw

レベル３

書式

#include <basic0. h>

void b_csw(n) ;
int n; /* on/off 指定 */

機能

カーソルスイッチの on/off の指定を行います。
n には次の値を指定します。

0 : off
1 : on

戻り値

戻り値はありません。

プログラム例

```
#include        <basic0.h>

main()
{
        char    a[4];

        b_init();
        b_inkeys(a);
        b_sprint("csw off");
        b_sprint(STRCRLF);
        b_csw(0);
        b_inkeys(a);
        b_sprint("csw on");
        b_sprint(STRCRLF);
        b_csw(1);
        b_inkeys(a);
        b_exit(0);
}
```

b_dateS

レベル３

書　　式

```
#include <basic0.h>

char *b_dateS(s) ;
char *s; /*  日付文字列格納領域へのポインタ */
```

機　　能

システムクロックの日付を“yy/mm/dd”の形式の文字列にして、sが指す領域に格納します。

戻 り 値

作成した文字列へのポインタを返します。

b_dayS

レベル 3

書　式

```
#include <basic0.h>

char *b_dayS(s);
char *s;  /* 曜日文字列格納領域へのポインタ */
```

機　能　システムクロックの曜日を漢字1文字（"日"、"月"、"火"、"水"、"木"、"金"、"土"）の文字列にして、sが指す領域に格納します。

戻り値　作成した文字列へのポインタを返します。

beep

レベル3

書　式　#include <basic0. h>

```
void beep( ) ;
```

機　能　ビープ音を鳴らします。

戻り値　戻り値はありません。

b_exit

レベル３

書式

```
#include <basic0.h>

void b_exit(val);
int val;  /*終了コード*/
```

機能

すべての環境をプロセスが起動したときの状態に戻して、親プロセスに戻ります。

オープンされているファイルはすべてクローズされます。

valの数値は、終了コードとして親プロセスにわたされます。

本関数を使用する場合は、この関数が呼び出される前にb_init関数が呼び出されていなければいけません。

戻り値

戻り値はありません。

b_fclose

レベル3

書　式

```
#include <basic.h>

int b_fclose(fp);
int fp;  /* 該当ファイルのファイル番号 */
```

機　能

fpで指定したファイルをクローズします。

戻り値

正常にクローズした場合は0を返します。
オープンしていないファイルをクローズしようとしたときは、−2を返します。
その他のエラーのときには、−1を返します。

b_fcloseall

レベル 3

書　　式	# include <basic. h> int b_fcloseall( ) ;
機　　能	b_fopen 関数でオープンされたファイルをすべてクローズします。
戻 り 値	クローズしたファイル数を返します。

b_feof

レベル3

書　式

```
#include <basic.h>

int b_feof(fp);
int fp;  /*  該当ファイルのファイル番号  */
```

機　能

fpで指定したファイルがエンドオブファイル（EOF）に達したか否かを調べます。

戻り値

エンドオブファイルでない場合は0を返します。
オープンしていないファイルを指定したときは、−2を返します。
エンドオブファイルの場合は、上記以外の値を返します。

b_fgetc

レベル3

書　式

```
#include <basic.h>

int b_fgetc(fp) ;
int fp; /* 該当ファイルのファイル番号 */
```

機　能

fpで指定したファイルから1文字読み込みます。

戻り値

正常に読み込んだ場合、そのコードを返します。
オープンしていないファイルを指定したときは、−2を返します。
その他のエラーのときには、−1を返します。

b_flprint, b_ilprint, b_slprint, b_tlprint

レベル3

書　　式

```
#include <basic0.h>

void b_flprint(dval);
void b_ilprint(ival);
void b_slprint(str);
void b_tlprint(tabs);

double dval;
int ival;
char *str;
int tabs;
```

機　　能

BASICからCに変換したときにLPRINT文は、LPRINTの引数によって次のそれぞれの関数に展開されます。

引数の型	関数
double	b_flprint（浮動小数点数）
int	b_ilprint（整数）
char *	b_slprint（文字列）

また、BASICのTAB関数を含んだLPRINT文は、b_tlprintに展開されます。
tabsはタブスキップの数です。
それぞれの引数をプリンタに出力します。

戻り値

戻り値はありません。

b_fopen

レベル３

書式

```
#include <basic. h>

int b_fopen(name, mode) ;
char *name; /* ファイル名格納領域へのポインタ */
char *mode; /* アクセスモード */
```

機能

nameが指すファイルをオープンします。
アクセスモードは、次の通りです。

mode	内容
"r"	読み込み用にオープン
"w"	書き出し用にオープン
"rw"	読み書き用にオープン
"c"	新規にオープン

戻り値

ファイル番号を返します。
エラーのときは負の数を返します。

参照関数

b_fwrites, b_fclose, b_fread

b_fprint

レベル３

書　式

```
#include <basic0.h>

void b_fprint(f);
double f;
```

機　能

浮動小数点数値ｆをテキスト画面に表示します。
なお、整数表現が可能な数値には小数点はつきません。

戻り値

戻り値はありません。

b_fputc

レベル3

書　　式	# include <basic. h> int b_fputc(ch, fp) ; char ch ; /* 出力する文字 */ int fp ; /* 該当ファイルのファイル番号 */
機　　能	文字chを、fpで指定したファイルに書き出します。
戻り値	書き出した文字コードを返します。 オープンしていないファイルを指定したときは、−2を返します。 その他のエラーのときには、−1を返します。

b_fread

レベル3

書　式

```
#include <basic.h>

int b_fread(buff, b_size, count, fp) ;
void *buff; /* 読み込み領域へのポインタ */
int b_size; /* 読み込む項目のサイズ */
int count; /* 読み込む項目数 */
int fp; /* 該当ファイルのファイル番号 */
```

機　能

fpで指定したファイルから、b_size×countバイト数分読み込み、buffが指すバッファへ格納します。

戻り値

実際に読み込んだ項目数を返します。
オープンしていないファイルを読もうとしたときは、−2を返します
その他のエラーのときには、−1を返します。

参照関数

b_fopen, b_fclose, b_fwite

b_freads

レベル3

書式

```
#include <basic.h>

int b_freads(buff, size, fp);
char *buff;  /* 読み込み領域へのポインタ */
int size;  /* 読み込み領域のバイト数 */
int fp;  /* 該当ファイルのファイル番号 */
```

機能

ファイルから1行分のデータをバッファbuffに読み込みます。
バッファがいっぱいになるか、EOF、改行をみつけると、読み込みを終了します。

戻り値

実際に読み込んだ文字数を返します。
オープンしていないファイルを読もうとしたときは、−2を返します。
その他のエラーのときには、−1を返します。

参照関数

b_fwrites, b_fopen, b_fclose

b_free

レベル３

書　式

```
#include <basic0.h>

int b_free ( ) ;
```

機　能　スタック領域の残りバイト数を調べます。

戻り値　スタック領域の残りバイト数を返します。

b_fseek

レベル 3

書　式

```
#include <basic.h>

int b_fseek(fp, ofs, md);
int fp;  /* ファイルポインタを移動するファイルのファイル番号 */
int ofs; /* オフセット /*
int md;  /* ファイルポインタの起点 */
```

機　能

fpで指定したファイルのファイルポインタを、mdを起点としてofsバイト数分移動します。

mdは、次の意味をもちます。

0：先頭を起点とする
1：現在位置を起点とする
2：最後を起点とする

戻り値

先頭からのオフセット値を返します。

オープンしていないファイルを指定したときは、−2を返します。

その他のエラーのときには、−1を返します。

b_fwrite

レベル3

書　　式

```
#include <basic.h>

int b_fwrite(buff, size, count, fp) ;
void *buff; /* 書き出し領域へのポインタ */
int size; /* 書き出す項目のサイズ */
int count; /* 書き出す項目数 */
int fp; /* 該当ファイルのファイル番号 */
```

機　　能

buffが指すバッファから、size×countバイト数分、fpで指定したファイルへデータを書き出します。

戻 り 値

実際に書き出した項目数を返します。
オープンしていないファイルに書こうとしたときは、−2を返します。
その他のエラーのときには、−1を返します。

b_fwrites　レベル3

書　式　# include <basic. h>

```
int b_fwrites(buff, fp) ;
char *buff ;  /*  書き出し領域へのポインタ */
int fp ;  /*  該当ファイルのファイル番号 */
```

機　能　バッファ buff に入っている文字列をファイルに書き出します。

戻り値　実際に書き込んだ文字数を返します。
オープンしていないファイルに書こうとしたときは、−2を返します。
その他のエラーのときには、−1を返します。

参照関数　b_freads, b_fopen, b_fwrite

bg_fill

レベル3

書式

```
#include <sprite.h>

int bg_fill(t_page, pattern) ;
char t_page ;  /*  テキストページ  */
int pattern ;   /*  データパターン  */
```

機能

t_pageで指定したバックグラウンドテキストページを、patternで指定したデータで埋めつくします。

t_pageにはテキストページを指定します（0～1）。

patternに指定する内容は次の通りです。

ビット31～16：常に0
ビット15　　：垂直反転
ビット14　　：水平反転
ビット12～13：常に0
ビット11～8：パレットブロック
ビット7～0　：PCGパターンコード

戻り値

以下に示す終了コードを返します。

0 = 正常終了
−1 = 画面モードエラー（画面サイズが768×512のときなど）

bg_get　レベル3

書式　#include <sprite. h>

```
long bg_get(t_page, xpos, ypos) ;
char t_page; /* テキストページ */
char xpos; /* X座標 */
char ypos; /* Y座標 */
```

機能　バックグラウンドテキストを読み出します。
t_pageにはテキストページを指定します（0～1）。
xposにはX座標、yposにはY座標を指定します（いずれも0～63）。

戻り値　バックグラウンドテキストのコードを返します。
返される内容は次の通りです。

ビット31～16：常に0
ビット15　　：垂直反転
ビット14　　：水平反転
ビット12～13：常に0
ビット11～8 ：パレットブロック
ビット7～0　：PCGパターンコード

ただし、－1が返された場合は画面モードエラーです。
（画面サイズが768×512のときなど）

参照関数　bg_put

bg_put

レベル3

書　式

```
#include <sprite.h>

int bg_put(t_page, xpos, ypos, pattern) ;
char t_page ;  /*  テキストページ  */
char xpos ;  /*  X座標  */
char ypos ;  /*  Y座標  */
int pattern ;   /*  データパターン  */
```

機　能

バックグラウンドテキストに書き込みます。
t_page にテキストページを指定します（0～1）。
xpos には X 座標、ypos には Y 座標を指定します（いずれも 0～63）。
pattern にはデータパターンを指定します。
pattern に指定する内容は次の通りです。

ビット 31～16：常に 0
ビット 15　　：垂直反転
ビット 14　　：水平反転
ビット 13～12：常に 0
ビット 11～8：パレットブロック
ビット 7～0　：PCG パターンコード

戻り値

以下に示す終了コードを返します。

0 ＝ 正常終了
−1 ＝ 画面モードエラー（画面サイズが 768×512 のときなど）

参照関数

bg_get

bg_scroll

レベル3

書式

```
#include <sprite.h>

int bg_scroll(bg_no, x, y) ;
char bg_no ; /* バックグラウンド番号 */
int x ; /* スクロールX座標 */
int y ; /* スクロールY座標 */
```

機能

bg_noで指定するバックグラウンド（0～1）に、x，yで示すスクロール座標を設定します。

戻り値

以下に示す終了コードを返します。

0 ＝ 正常終了
−1 ＝ 画面モードエラー（画面サイズが768×512のときなど）

bg_set

レベル3

書　式

```
#include <sprite.h>

int bg_set(bg_no, t_page, disp) ;
char bg_no ;  /*  バックグラウンド番号  */
int t_page ;  /*  テキストページ  */
int disp ;  /*  表示モード  */
```

機　能

バックグラウンドテキストの設定を行います。
bg_no にはバックグラウンド番号を指定します（0～1）。
t_page にはテキストページを指定します（0～1）。
disp には表示モードを指定します。
1を指定した場合には表示を行い、0を指定した場合には表示は行いません。

戻り値

以下に示す終了コードを返します。

0 ＝ 正常終了
−1 ＝ 画面モードエラー（画面サイズが768×512のときなど）

bg_stat

レベル3

書式

```
#include <sprite.h>

int bg_stat(bg_no, sel_mode) ;
char bg_no ;  /* バックグラウンド番号  */
char sel_mode ;  /* モード  */
```

機能

指定したバックグラウンドの状態を調べます。
bg_no にはバックグラウンド番号を指定します (0～1)。
sel_mode には調べたい内容を指定します。

0 ：X座標
1 ：Y座標
2 ：テキストページ (0～1)
3 ：表示状態 (0＝表示していない、1＝表示している)

戻り値

sel_mode に合わせて、それぞの値を返します。
ただし、−1が返された場合は画面モードエラーです。
(画面サイズが768×512のときなど)

b_hexS

レベル３

書　式

```
#include <basic.h>

char *b_hexS (s, n);
char *s;  /* 文字列格納領域へのポインタ */
int n;  /* 数値 */
```

機　能

数値nを16進数表現の文字列に変換し、sが指す領域へ格納します。
sが指す領域は、最低10バイト以上なければなりません。

戻り値

変換結果の文字列へのポインタを返します。

b_init

レベル3

書　式　

```
#include <basic0.h>

void b_init( ) ;
```

機　能　BASICからCへ変換したときにBASICと同じ状態で動作するための初期化を行います。
初期化の内容は次の通りです。

- ●現在の画面モードの退避
- ●ファンクションキーの内容の退避
- ● BASIC 用ファンクションキーの設定
- ●アボート処理の登録
- ●乱数の初期化
- ●カーソルの非表示

戻り値　戻り値はありません。

b_inkey0

レベル3

書　式

```
#include <basic0.h>

char *b_inkey0(s);
char *s;  /* キーコード格納領域へのポインタ*/
```

機　能

キー入力待ちを行わずにキー入力を行い、入力した文字をｓが指す領域に格納します。
入力した文字には '¥0' が付加されますので、ｓが指す領域は２バイト以上必要です。

戻り値

入力した文字へのポインタを返します。

b_inkeyS　レベル3

書　式　# include <basic0. h>

```
char *b_inkeyS(s) ;
char *s; /* キーコード格納領域へのポインタ */
```

機　能　キーの入力を待ってキー入力を行い、入力した文字をsが指す領域に格納します。

入力した文字には '¥0' が付加されますので、sが指す領域は2バイト以上必要です。

戻り値　入力した文字へのポインタを返します。

b_input

レベル３

書　　式

```
#include <basic0.h>

void b_input (prmpt, [type, varadr] ・・・, −1) ;
char *prmpt ;  /* プロンプト文字列へのポインタ */
int type ;  /*  変換書式  */
void *varadr ;  /* 読み込み領域のアドレス */
```

機　　能

prmptが指すプロンプト文字列を表示して入力を待ち、typeに従って変換し、varadrから始まる領域に値を格納します。
文字列の場合は、typeで示す長さ−1までが有効な入力となります。
入力する変数が複数個の場合には入力時に“,”で区切ります。
入力データ中に“,”や“ ””を使うことはできません。
typeには、次の値を指定します。

−1	:	終りを表します（varadrは指定しません）
1～257	:	文字列の長さ（文字列の最後の¥0を含む長さ）
0x201	:	文字型（unsigned char）
0x204	:	整数型（int）
0x208	:	浮動小数点型（double）

typeとvaradrの型は、以下のように対応していなければなりません。

type		varadrの型
1～257	:	unsigned char *
0x201	:	unsigned char *
0x204	:	int *
0x208	:	double *

戻 り 値

戻り値はありません。

b_int

レベル3

書式

```
#include <basic0.h>

int b_int(f) ;
double f;  /*  変換される浮動小数点数  */
```

機能 浮動小数点数を整数型に変換します。

戻り値 変換した数値を返します。

b_iprint

レベル3

書式

```
#include <basic0.h>

void b_iprint(val);
int val;  /* 整数値 */
```

機能

BASICからCに変換したときにPRINT文の引数が整数のとき、b_iprintに展開されます。
整数を画面に出力します。

戻り値

戻り値はありません。

b_itoa

レベル3

書式

```
#include <basic.h>

char *b_itoa(str, val);
char *str;  /* 文字列格納領域へのポインタ */
int val;  /* 整数値 */
```

機能

BASICからCに変換したときにITOA関数は、b_itoaに展開されます。
数値valを文字列に変換し、strが指す領域へ格納します。
strが指す領域には、最低13バイト以上の領域が必要です。

戻り値

変換結果の文字列へのポインタを返します。

bldmem, allmem　レベル2

書　式

```
#include <stdlib.h>

int bldmem(n) ;  /*  メモリブロックの確保  */
int n;  /*  確保する領域のサイズ（キロバイト単位）*/

int allmem( ) ;  /* 使用可能な全メモリ領域の確保  */
```

機　能

メモリ領域の確保を行います。
bldmem は、メモリブロックをキロバイト単位で確保します。
n に 0 を指定すると、使用可能な全メモリ領域を確保します。
allmem は、使用可能な全メモリ領域の確保を行います。
これらの関数によって確保した領域は、getmem 関数などを用いて割りあて、使用することができます。

戻り値

処理が正常に終了した場合には 0 を返し、領域が確保できないときには−1 を返します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        int     n;

        if (allmem() == 0)
                printf("全メモリを確保しました。¥n");
        else {
                printf("メモリの確保に失敗しました。¥n");
                exit(1);
        }
        rstmem();
        printf("すべてのメモリを解放しました¥n");

        for (n = 0; n < 1000; n++) {
                printf("%dKbyte: ", n);
                if (bldmem(n) == 0) {
                        printf("確保できました。");
                        rstmem();
                        printf(": 全メモリを解放しました¥n");
                }
                else {
                        printf("確保できませんでした。¥n");
                        break;
                }
        }
}
```

b_leftS

レベル 3

書　　式

```
#include <basic.h>

char *b_leftS(s, s1, i);
char *s;  /* 結果文字列格納領域へのポインタ */
char *s1; /* 文字列 */
int i;    /* 文字数 */
```

機　　能

文字列 s1 の左端から i 文字分とり出し、s が指す領域へ格納します。
s が指す領域は、最低 256 バイト以上なければなりません。
また、s1 が指す文字列の長さは、255 以下でなければなりません。

戻 り 値

とり出した文字列 s へのポインタを返します。

b_linput

レベル3

書式

```
#include <basic0.h>

void b_linput(s, size) ;
char *s; /* 入力文字列の格納領域へのポインタ */
int size; /* 入力する領域のバイト数 */
```

機能

キーボードからの入力を、最大size−1文字数、sが指す領域に格納します。
sizeには、文字列の終りを示す'¥0'を含めた領域のバイト数を指定します。
文字列の最後には'¥0'が付加されます。
したがって、sが指す領域には最大でsize−1の文字が入力されます。
ただし、2行にわたっての入力はできません。

戻り値

戻り値はありません。

b_midS　レベル 3

書　　式

```
#include <basic.h>

char *b_midS(s, s1, bgn, count);
char *s;  /* 結果文字列格納領域へのポインタ */
char *s1;  /* 文字列 */
int bgn;  /* 開始位置 */
int count;  /* 文字数 */
```

機　　能

文字列 s1 の bgn 文字目から count 文字だけをとり出し、s が指す領域へ格納します。

s が指す領域は、最低 256 バイト以上なければなりません。

また、s1 が指す文字列の長さは、255 以下でなければなりません。

戻 り 値

とり出した文字列 s へのポインタを返します。

b_mirrorS

レベル３

書　　式

```
#include <basic.h>

char *b_mirrorS(s, s1);
char *s;  /* 結果文字列格納領域へのポインタ */
char *s1; /*  文字列 */
```

機　　能

文字列s1の並びを前後逆にして、sが指す領域へ格納します。
sが指す領域は、最低256以上バイトなければなりません。
また、s1が指す文字列の長さは、255以下でなければなりません。

戻 り 値

並びかえた文字列sへのポインタを返します。

b_octS

レベル３

書式	

```
#include <basic.h>

char *b_octS(s, n) ;
char *s; /* 文字列格納領域へのポインタ */
int n; /* 数値 */
```

機能　数値nを８進数表現の文字列に変換し、sが指す領域へ格納します。
sが指す領域は最低13バイト以上なければなりません。

戻り値　変換結果の文字列へのポインタを返します。

box

レベル3

書式

```
#include <graph.h>

void box (x1, y1, x2, y2, p, ls) ;
int x1 ; /*  始点X座標  */
int y1 ; /*  始点Y座標  */
int x2 ; /*  終点X座標  */
int y2 ; /*  終点Y座標  */
int p ;  /*  パレットコード  */
int ls ; /*  ラインスタイル  */
```

機能

グラフィック画面に長方形を描画します。

x1には始点X座標を指定します（−32768～32767）。
y1には始点Y座標を指定します（−32768～32767）。
x2には終点X座標を指定します（−32768～32767）。
y2には終点Y座標を指定します（−32768～32767）。
pにはパレットコードを指定します（0～65535）。
画面モードにより最大値が異なります。
lsにはラインスタイルを指定します（0～65535）。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdlib.h>
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i;
        int     x1, x2, y1, y2;
        int     pc;
        int     ls;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        for (i = 0; i < 500; i++) {
                x1 = rand() % 768;
                y1 = rand() % 512;
                x2 = rand() % 768;
                y2 = rand() % 512;
                pc = rand() % 16;
                ls = rand() * 2 + 1;
                box(x1, y1, x2, y2, pc, ls);
        }
        b_exit(0);
}
```

b_pi

レベル3

書　式

```
#include <basic0.h>

double b_pi(f);
double f;  /* 倍率 */
```

機　能　円周率πをf倍した値を求めます。

戻り値　$f \times \pi$の値を返します。

参照関数　pi

b_rightS

レベル 3

書式

```
#include <basic. h>

char *b_rightS(s, s1, i) ;
char *s; /* 結果文字列格納領域へのポインタ */
char *s1; /*  文字列  */
int i; /* 文字数 */
```

機能

文字列 s1 の右端から i 文字分とり出し、s が指す領域へ格納します。
s が指す領域は、最低 256 バイト以上なければなりません。
また、s1 が指す文字列の長さは、255 以下でなければなりません。

戻り値

とり出した文字列 s へのポインタを返します。

bsearch

レベル2

書　式

```
#include <stdlib.h>

void  *bsearch(key, base, num, width, compare) ;
void  *key ;  /*  サーチするデータを指すポインタ  */
void  *base ;  /*  配列を指すポインタ  */
size_t num, width ;  /*  配列要素の数、配列要素のサイズ  */
int  (*compare)( ) ;  /* compareルーチンを指すポインタ  */
```

機　能

各要素がwidthバイトで、要素数がnum個の配列をバイナリサーチします。
baseは配列を指すポインタでkeyはサーチする値へのポインタです。
なお、この配列はあらかじめ昇順にソートされていなければなりません。
compareは、サーチする値と配列の要素を比較し、その大小関係を返すユーザーが用意するルーチンを指すポインタです。
bsearchは、サーチの過程で、必要に応じてcompareが指す関数を呼び出します。
compareは2つの引数を受けとり、1番目の引数はサーチする値へのポインタ、2番目の引数は配列の要素へのポインタが渡されるので、その2つを比較して、次の値のうちいずれかを返す関数でなければなりません。

compareの戻り値	内　容
負	サーチする値が要素よりも小さい
0	サーチする値と要素は等しい
正	サーチする値が要素よりも大きい

戻り値

最初に見つかった配列要素へのポインタを返します。
見つからなかった場合はNULLを返します。

参照関数

qsort

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>

#define MAX     12

typedef struct {
        char    *key;
        char    *name1;
        char    *name2;
} ELM;

/* This table must be sorted by alphabetical */
static ELM      table[MAX] = {
        {"Aqr", "Aquarius", "水瓶"},
        {"Ari", "Aries", "牡羊"},
        {"Cap", "Capricornus", "山羊"},
        {"Cnc", "Cancer", "蟹"},
        {"Gem", "Gemini", "双子"},
        {"Leo", "Leo", "獅子"},
        {"Lib", "Libra", "天秤"},
        {"Psc", "Pisces", "魚"},
        {"Sco", "Scorpius", "蠍"},
        {"Sgr", "Sagittarius", "射手"},
        {"Tau", "Taurus", "牡牛"},
        {"Vir", "Virgo", "乙女"}
};


main(int argc, char *argv[])
{
        int     compare(char **, ELM *);
        char    *key;
        ELM     *p;

        if (argc < 2)
                exit(1);
        key = argv[1];
        if ((p = (ELM *)bsearch(&key, (char *)table,
                  MAX, sizeof(ELM), compare)) == NULL)
                printf("not found¥n");
        else
                printf("%s: %s¥n", p->name1, p->name2);
}


int     compare(char **key, ELM *elm)
{
        return (strcmp(*key, elm->key));
}
```

b_setdate

レベル3

書　式

```
#include <basic0.h>

void b_setdate(s);
char *s; /* 日付を表す文字列へのポインタ */
```

機　能

システムクロックの日付を設定します。
sが指す文字列は"87/09/30"のような形式で指定してください。

戻り値

戻り値はありません。

b_settime

レベル 3

書　式

```
#include <basic0.h>

void b_settime(s) ;
char *s; /* 時刻を表す文字列へのポインタ */
```

機　能

システムクロックの時刻を設定します。
sが指す文字列は"21:56:30"のような形式で指定してください。

戻り値

戻り値はありません。

b_spaceS

レベル3

書　式

```
#include <basic0. h>

char *b_spaceS(s, n) ;
char *s; /* 作成した文字列を格納する領域へのポインタ */
int n; /* 文字数 */
```

機　能

sの指す領域に、n文字の空白文字からなる文字列を作成します。
nに256以上の値を指定すると、nは0とみなされます。
nに0を指定すると、長さが0の文字列（空文字列）を作成します。
sが指す領域は、最低256バイト以上なければなりません。

戻り値

作成した文字列sへのポインタを返します。

b_sprint

レベル3

書　式

```
#include <basic0.h>

void b_sprint(s);
char *s; /* 表示文字列へのポインタ */
```

機　能　sが指す文字列をテキスト画面に表示します。

戻り値　戻り値はありません。

b_stradd

レベル3

書　式

```
#include <basic0. h>

char *b_stradd(s, s1, s2, ・・・・, -1) ;
char *s; /* 連結した文字列を格納する領域へのポインタ */
char *s1; /* 文字列へのポインタ */
char *s2; /* 文字列へのポインタ */
```

機　能

複数の文字列を連結した文字列を作ります。
連結した文字列の長さが255文字以上になる場合は、255文字まで連結し、それ以降の文字は連結されません。
sが指す領域は、最低256バイト以上なければなりません。
引き数の最後には、これ以上連結する文字列がないことを表す−1を指定します。

戻り値

連結した文字列へのポインタを返します。

b_strchr, b_strrchr, b_strtok

レベル3

書式

```
#include <basic.h>

int b_strchr(str, ch);
int b_strrchr(str, ch);
char *b_strtok(str1, str2);

char *str, *str1, *str2;
char ch;
```

機能

b_strchr は、str の中の最初にみつかった ch の位置を求めます。
b_strrchr は、str の中の最後にみつかった ch の位置を求めます。
b_strtok は検索文字集合 str2 に含まれる文字のいずれか1つの文字を str1 から探し、みつかった位置までの文字列を求めます。
b_strtok の2回目以降の呼び出しで、str1 に長さ0の文字列を指定すると、前回の続きから検索します。
このとき、検索文字集合は前回と異なる文字集合を指定してもかまいません。

BASIC の strchr, strrchr, strtok がそれぞれ展開されたもので、C の標準ライブラリにある strchr, strrchr, strtok とは仕様が異なります。
b_strtok は、内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。
また、str1 に長さが0でない文字列を指定すると、以前に返された文字列は失われます。

戻り値

b_strchr, b_strrchr は、ch の見つかった位置を返します。
見つからないときは−1を返します。
b_strtok は、見つかった文字列へのポインタを返します。
見つからないときは、長さが0の文字列へのポインタを返します。

b_strcmp, b_strncpy　レベル3

書　式

```
#include <basic0.h>

int b_strcmp(str1, op, str2) ;
void b_strncpy(num, str1, str2) ;

char *str1, *str2;
int op, num;
```

機　能

b_strcmp は、2つの文字列 str1, str2 を比較し、指定された比較条件が成り立つかどうか調べます。

BASIC の文字列比較（if a$ <"abc" など）が展開されたもので、C のライブラリにある strcmp とは仕様が異なります。

<=や< などの記号は、op にわたされます。

OP に指定できる条件は以下のとおりです。

'= '	(0x3d20)	等しい
'=='	(0x3d3d)	等しい
'<>'	(0x3c3e)	等しくない
'><'	(0x3e3c)	等しくない
'!='	(0x213d)	等しくない
'>='	(0x3e3d)	大きいか等しい
'=>'	(0x3c3e)	大きいか等しい
'<='	(0x3c3d)	小さいか等しい
'=<'	(0x3d3c)	小さいか等しい
'< '	(0x3c20)	小さい
'> '	(0x3e20)	大きい

b_strncpy は、コピーする文字数+1を num に指定して、文字列 str2 を文字列 str1 にコピーします。

str1 が指す領域は、num で示すバイト数の領域が必要です。

num には、文字列の終りを示す '¥0' を含めた文字数を指定するので、通常は sizeof（str1）のように指定します。

num に 0 を指定すると、str1 の文字列は変化しません。

また、1を指定すると、str1 の文字列の長さが 0 になります。

BASIC の文字列代入（a$="abc" など）が展開されたもので、C の標準ライブラリにある strncpy とは仕様が異なります。

戻り値　b_strcmp は、比較の結果、op に指定された条件が成立した場合は 0 以外を返し、不成立の場合は 0 を返します。

結果	戻り値
成り立つ	0 以外
成り立たない	0

b_strncpy は、戻り値はありません。

b_strfS

レベル3

書　式

```
#include <basic0.h>

char *b_strfS(s, f);
char *s;  /* 文字列格納領域へのポインタ */
double f;  /*  浮動小数点数値 */
```

機　能

浮動小数点値 f を文字列に変換し、s が指す領域に格納します。
変換する数値が負の数のときには、負の符号がつきます。
s が指す領域には、最低 23 バイト以上の領域が必要です。

戻り値

変換した文字列 s へのポインタを返します。

b_striS

レベル3

書式 #include <basic0.h>

```
char *b_striS(s, n);
char *s; /* 文字列格納領域へのポインタ */
int n; /* 整数値 */
```

機能 整数値nを文字列に変換し、sが指す領域に格納します。
変換する数値が負の数のときには、負の符号がつきます。
sが指す領域には、最低13バイト以上の領域が必要です。

戻り値 変換した文字列sへのポインタを返します。

b_stringS

レベル3

書式 #include <basic. h>

```
char *b_stringS(s, i, st);
char *s; /* 文字列格納領域へのポインタ */
int i; /* 並べる文字数 */
char *st; /* 該当文字格納領域へのポインタ */
```

機能 文字列stで指定した文字列の先頭1文字を、i文字分並べた文字列を作成し、sが指す領域へ格納します。
BASICのstring$が展開されたものです。
iに256以上を指定すると、iは0とみなされます。
iに0を指定すると、長さが0の文字列（空文字列）を作成します。
sが指す領域は少なくとも256バイトなければなりません。

戻り値 作成した文字列sへのポインタを返します。

b_timeS

レベル3

書　式

```
#include <basic0.h>

char *b_timeS(s);
char *s;  /* 時刻設定文字列へのポインタ */
```

機　能　システムクロックの時刻を"tt:mm:ss"の形式の文字列にして、sが指す領域に格納します。

戻り値　作成した文字列へのポインタを返します。

b_tpalet

レベル3

書　式

```
#include <basic0.h>

void b_tpalet(p0, p1, p2, p3) ;
int p0, p1, p2, p3 ;  /*  パレット属性 */
```

機　能

テキストのパレット色を設定します。
p0～p3にはパレットコード0～3に対する色コードを指定します。

戻り値

戻り値はありません。

b_tprint　レベル3

書　式

```
#include <basic0.h>

void b_tprint(n) ;
int n;  /* タブスキップ数  */
```

機　能

カーソルの桁の位置がnより小さければカーソル位置がnになるまでスペースを表示します。

nより大きいか等しければ何も行いません。

戻り値

戻り値はありません。

cabs

レベル2

書　式

```
#include <math.h>

double cabs(z);
struct complex  *z;
```

機　能

複素数の絶対値を求めます。

zは、複素数を格納してあるcomplex型構造体へのポインタを指定します。

```
struct complex {
                double x, y;
                };
```

また、cabsは下の式と同じ結果を与えます。

```
sqrt (z->x*z->x+z->y*z->y)
```

戻り値

複素数の絶対値を返します。

参照関数

abs, fabs, labs

calloc

レベル２

書　式

```
#include <stdlib.h>

void  *calloc(n, size);
size_t n;  /*  配列要素の個数  */
size_t size;  /*  配列要素のサイズ  */
```

機　能

各要素の大きさがsizeバイトで、要素数がn個の配列のための領域を割りつけます。
確保された配列は0で初期化されます。

戻り値

割りつけた領域を指すポインタを返します。
領域が確保できなかったときには、NULLを返します。

参照関数

free, malloc, realloc

ceil

レベル2

書　　式

```
#include <math.h>

double ceil(x) ;
double x ;
```

機　　能　xより小さくない最小の整数を求めます。

戻 り 値　doubleとしての整数を返します。

参照関数　floor, modf

cgets　レベル2

書式　#include <conio. h>

```
char  *cgets(str) ;
char  *str; /*  文字列格納領域へのポインタ  */
```

機能　コンソールから文字列を読み込み、その長さと文字列をを配列strに格納します。

strが指す領域は、読み込まれる文字列の最大長+3バイトなければなりません。

str [0]：読み込まれる文字列の最大長
str [1]：読み込まれた文字列の実際の長さ
str [2]：格納した文字列の先頭位置

cgetsは、復帰改行を読み込むか指定された文字数str [0] に達するまでコンソールから入力される文字を読み続け、str [2] から順に格納します。
復帰改行を読み込むと、それをヌル文字（'¥0'）に置き換えて格納します。
読み込みが終了すると、str [1] に実際の文字列の長さを格納します。

戻り値　読み込んだ文字列へのポインタを返します。

参照関数　fgets, gets

呼び出しの前にstr [0] に読み込まれる文字列の最大長を代入しておかなければなりません。

chdir

レベル2

書　式

```
#include <direct.h>

int chdir(pathname);
char  *pathname;  /*  変更先ディレクトリ名へのポインタ  */
```

機　能

カレントディレクトリをpathnameによって指定したディレクトリに変更します。

pathnameが示すディレクトリは既存のものでなければなりません。

戻り値

正常終了ならば正の数（0も）を返します。

エラーならば−1を返します。

また、エラーが発生した場合、errnoと_doserrnoにそれぞれCのエラー番号とOSのエラー番号をセットします。

参照関数

mkdir, rmdir

chgfa

レベル2

書　式

```
#include <io. h>

int chgfa(pathname, atr) ;
char  *pathname ;  /*  属性を変更するファイルのパス名へのポインタ  */
int  atr ;  /*  ファイルの属性  */
```

機　能

pathnameで指定したファイルの属性を、atrで指定した属性に変更します。
atrで指定できる属性は以下の通りです。

値	属性	指定方法
0x01	読み込み専用	組合せで0x00～0x07まで指定できます。
0x02	隠しファイル	
0x04	システムファイル	
0x08	ボリューム名	3つの中から1つだけ指定し、さらに上記の0x00～0x07との加算した値を設定します。
0x10	ディレクトリ	
0x20	通常ファイル	

戻り値

正常終了ならば0を返します。
エラーならば−1を返します。

プログラム例

```
#include        <stdio.h>
#include        <io.h>

#define A_RDO           0x01    /* 読み込み専用 */
#define A_HDN           0x02    /* 隠しファイル */
#define A_SYS           0x04    /* システムファイル */
#define A_VOL           0x08    /* Volumeラベル */
#define A_DIR           0x10    /* ディレクトリ */
#define A_FIL           0x20    /* 通常ファイル */


main()
{
        int     r;

        r = chgfa("¥¥CONFIG.SYS", A_RDO|A_FIL);

        if (r == 0)
                puts("成功しました。");
        else
                puts("失敗しました。");
}
```

chgft

レベル2

書　式

```
#include <io. h>

int  chgft(file, datetime) ;
int  file; /*  作成日時を変更するファイルのファイルハンドル  */
int  datetime ;   /*  ファイルの作成日時  */
```

機　能

fileで指定したファイルの作成日時を、dateimteで指定した作成日時に変更します。

datetimeのフォーマットは以下の通りです。

ビット31～25　：1980年からの相対年数(0～99)
ビット24～21　：月(1～12)
ビット20～16　：日(1～31)
ビット15～11　：時(0～23)
ビット10～5　：分(0～59)
ビット4～0　：秒/2(0～29)

戻り値

正常終了ならば0を返します。
エラーならば−1を返します。

プログラム例

```
#include        <stdio.h>
#include        <time.h>
#include        <io.h>
#include        <fcntl.h>
#include        <stat.h>


main()
{
        struct tm       *mp;
        time_t  tp;
        long    datetime;
        int     fn;
        char    pktime[6];

        tp = time(NULL);
        mp = localtime(&tp);
        pktime[0] = mp->tm_year - 80;
        pktime[1] = mp->tm_mon + 1;
        pktime[2] = mp->tm_mday;
        pktime[3] = mp->tm_hour;
        pktime[4] = mp->tm_min;
        pktime[5] = mp->tm_sec;
        datetime = ftpack(pktime);

        if ((fn = open("¥¥CONFIG.SYS", O_RDWR)) == -1)
                puts("ファイルがオープンできません。");
        else {
                if (chgft(fn, datetime) == 0)
                        puts("成功しました。");
                else
                        puts("失敗しました。");
                close(fn);
        }
}
```

child

レベル3

書式

#include <basic. h>

int child(pathname) ;
char *pathname; /*　実行するファイルのパス名へのポインタ　*/

機能

pathnameで指定した実行ファイルをメモリ上にロードし、子プロセスとして起動します。

戻り値

正常終了ならば子プロセスの終了コードを返します。
エラーならば負の数を返します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <basic.h>

main()
{
        if (child("foo.X") < 0) {
                puts("エラー");
                exit(1);
        }
        else
                exit(0);
}
```

chkml

レベル2

書　式　# include <stdlib. h>

long chkml( ) ;

機　能　未使用メモリブロックの最大サイズをバイト数で求めます。

戻り値　メモリサイズがバイト数で返ります。
メモリに空きがない場合は0が返ります。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

main()
{
        long    size;

        size = chkml();
        printf("最大のブロックは %ldバイト¥n",size);
}
```

chmod

レベル2

書　式

```
#include <stat.h>
#include <io.h>

int chmod(pathname, pmode) ;
char  *pathname ; /*  モードを変更するファイルのパス名  */
int pmode ; /*  アクセスモード  */
```

機　能

pathnameで指定したファイルのアクセスモードをpmodeで指定したアクセスモードに変更します。
pmodeは、stat.hで定義されている定数S_IWRITE、S_IREADか、またはその2つを論理和した値を指定します。
他の値を指定すると、アクセスモードは変更しません。
引数pmodeに指定できる値とその内容を以下に示します。

pmodeの値	内　容
S_IWRITE	書き込み可
S_IREAD	読み込み可
S_IREAD | S_IWRITE	読み書きともに可

書き込み可でないファイルは、読み出し専用になります。
Human68kでは、S_IWRITEモードとS_IREAD | S_IWRITEモードは同じです。

戻り値

アクセスモードを変更することができた場合は0を返します。
エラーが起きた場合は−1を返し、errnoにエラー番号がセットされます。
ENOENTは指定したファイルを発見できなかったことを表しています。

参照関数

access, creat, fstat, open, stat

chsize　レベル２

書　式

```
#include <io.h>

int chsize(handle, size);
int handle;  /*  該当ファイルのファイルハンドル  */
long size;  /*  ファイルの変更サイズ  */
```

機　能

handleで指定したファイルの大きさをsizeで指定した大きさに変更します
この場合、ファイルはオープンされており、かつ書き込み可でなければなりません。
ファイルを大きくした場合にはヌル文字（'¥0'）がファイルの最後に大きくしたサイズ分だけ追加されます。
また、ファイルを小さくした場合は、ファイルの最後から小さくした分のデータが削除されます。

戻り値

ファイルの大きさを変更できた場合は0を返します。
変更できなかった場合は−1を返し、errnoに以下の値がセットされます。

errnoの値	内　容
EACCES	指定したファイルハンドルは読み出し専用
EBADF	ファイルハンドルが無効
ENOSPC	デバイスに空きがない

参照関数

close, creat, open

circle

レベル3

書　式

```
#include <graph.h>

void circle (x, y, r, p, s, e, hv);
int x;  /*  中心のX座標  */
int y;  /*  中心のY座標  */
int r;  /*  半径  */
int p;  /*  パレットコード  */
int s;  /*  開始角度  */
int e;  /*  終了角度  */
int hv;  /*  偏平率  */
```

機　能

グラフィック画面に円、円弧．描画します。

xには中心のX座標を指定します（−32768～32767)。
yには中心のY座標を指定します（−32768～32767)。
rには半径を指定します（0～32767)。
pにはパレットコードを指定します（0～65535)。
画面モードにより最大値が異なります。
sには円の開始角度を指定します（−360～360)。
開始角度に負の数を指定すると中心点から開始点までの直線を描画します。
eには円の終了角度を指定します（−360～360)。
終了角度に負の数を指定すると中心点から終了点までの直線を描画します。
hvには偏平率を指定します（0～65535)。
0～255で縦の比率がhv／256になります。
257～65535で、横の比率が256／hvになります。
256で真円になります。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdlib.h>
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i;
        int     x, y;
        int     r;
        int     pc;
        int     st, et;
        int     hv;

        b_init();
        screen(2, 0, 1, 1);
        for (i = 0; i < 500; i++) {
                x = rand() % 768;
                y = rand() % 512;
                r = rand() % 200;
                pc = rand() % 16;
                st = 0;
                et = 360;
                hv = 256;
                circle(x, y, r, pc, st, et, hv);
        }
        wipe();
        for (i = 0; i < 500; i++) {
                x = rand() % 768;
                y = rand() % 512;
                r = rand() % 200;
                pc = rand() % 16;
                st = 60;
                et = 30;
                hv = 256;
                circle(x, y, r, pc, st, et, hv);
        }
        wipe();
        for (i = 0; i < 500; i++) {
                x = rand() % 768;
                y = rand() % 512;
                r = 150;
                pc = rand() % 16;
                st = 0;
                et = 360;
                hv = rand() % 256;
                circle(x, y, r, pc, st, et, hv);
        }
        b_exit(0);
}
```

clearerr, clrerr

レベル2

書　式

```
#include <stdio.h>

void clearerr(stream);
void clrerr(stream);
FILE *stream; /* FILE構造体へのポインタ */
```

機　能

指定されたストリームのエラーインジケータおよびエンドオブファイル(EOF)インジケータをリセットします。
一度エラーインジケータがストリームにセットされると、clearerrまたはrewind関数を呼ぶまでそのストリームに対する処理はエラーになります。

戻り値

戻り値はありません。

参照関数

feof, ferror

clock　レベル 2

書　式　#include <time. h>

clock_t clock( );

機　能　プロセスの使用時間を得ます。

clock は、プロセスの起動からの経過時間を返します。

ある処理の最初と最後で clock を呼び出し、その２つの時点での戻り値の差を求める事で、その処理に要した時間を測定する事ができます。

その時間を秒単位で表すには、マクロ CLK_TCK または CLOCKS_PAR_SEC で割る必要があります。

戻り値　プロセスの起動時からの経過時間を返します。

ただし、プロセスの経過時間が得られない場合や、その値が表現可能な範囲にない場合は、−1 を返します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <time.h>

#define N_MIN   1L
#define N_MAX   1000000L


main()
{
        clock_t start;
        clock_t end;
        double  sec;
        long    sum;
        long    l;

        if ((start = clock()) == -1) {
                printf("process起動時からの時間が得られません。¥n");
                exit(1);
        }
        start = clock();
        sum = 0;
        for (l = N_MIN; l <= N_MAX; l++)
                sum += l;
        end = clock();

        sec = (double)(end - start) / (double)CLK_TCK;
        printf("%ldから%ldまでの和: %ld¥n", N_MIN, N_MAX, sum);
        printf("処理に要した時間: %g秒¥n", sec);
}
```

close

レベル2

書　式

#include <io. h>

int close(handle) ;
int handle ; /* クローズするファイルのファイルハンドル */

機　能

handleで指定されたファイルをクローズします。

戻り値

ファイルが正常にクローズされた場合は0を返します。
クローズされなかった場合は−1を返し、errnoにEBADFがセットされます。
これはhandleが正しくないことを示しています。

参照関数

creat, open

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <io.h>
#include        <stat.h>


main(int argc, char *argv[])
{
        char    *file;
        int     fn;

        file = (argc > 1) ? argv[1] : "file";

        if ((fn = open(file, O_RDONLY)) == -1) {
                printf("%sをopenできません。¥n", file);
                exit(1);
        }
        printf("%sをopenしました。¥n", file);

        if (close(fn) == -1) {
                printf("%sをcloseできません。¥n", file);
                exit(1);
        }
        printf("%sをcloseしました。¥n", file);

        exit(0);
}
```

cls

レベル３

書　式　#include <basic0. h>

```
void cls( ) ;
```

機　能　テキスト画面をクリアします
なお、b_sprint（"¥x0c"）を実行した場合も同じ結果が得られます。

戻り値　戻り値はありません。

color

レベル3

書　式

```
#include <basic0.h>

void color(n);
int n;  /*  文字属性  */
```

機　能

テキスト画面の文字属性を設定します。
nには次の内容を指定します。

0 ＝ 黒	8 ＝ 黒
1 ＝ 水色	9 ＝ 水色のリバース
2 ＝ 黄色	10 ＝ 黄色のリバース
3 ＝ 白	11 ＝ 白のリバース
4 ＝ 黒	12 ＝ 黒
5 ＝ 水色の強調	13 ＝ 水色の強調・リバース
6 ＝ 黄色の強調	14 ＝ 黄色の強調・リバース
7 ＝ 白の強調	15 ＝ 白の強調・リバース

なお、テキストのパレットはb_tpalet関数で指定します。

戻り値

戻り値はありません。

参照関数

b_tpalet

console　レベル3

書式

```
#include <basic0.h>

void console(ys, yl, fkey) ;
int ys;  /*  スクロール開始行  */
int yl;  /*  スクロール行数  */
int fkey;  /*  ファンクションキー  */
```

機能

スクロール行の範囲を ys 行から yl 行分とします。
本関数実行後、カーソルはホームポジションに移動します。
fkey はファンクションキー表示の on/off の指定です。
0 を指定すると表示なし、1 を指定すると表示ありになります。

戻り値

戻り値はありません。

contrast

レベル 3

書　式　# include <graph. h>

void contrast(i) ;
int i;　/*　コントラスト　*/

機　能　画面のコントラストを 0 から 15 の値で指定します。

戻り値　戻り値はありません。

プログラム例

```
#include        <basic0.h>
#include        <graph.h>

#define N_CONT          16


main()
{
        int     i;
        int     j;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        symbol(300, 200, "x68000", 2, 2, 2, 15, 0);
        for (i = 0; i < N_CONT; i++) {
                contrast(N_CONT - i - 1);
                for (j = 0; j < 5000; j++)
                        ;
        }
        b_exit(0);
}
```

cos, cosh　レベル2

書　式　# include <math. h>

double cos(x) ;
double cosh(x) ;
double x ;

機　能　cosはxのコサインを、coshはxのハイパボリック（双曲線）コサインを計算します。

戻り値　cosはxのコサインを返します。
coshはxのハイパボリックコサインを返します。
引数が不正な値の場合はerrnoにEDOMがセットされ返り値は不定になります。
オーバーフローが発生した場合はerrnoにERANGEがセットされ返り値は符号が付いたHUGE_VALになります。
またエラー処理は、matherr関数を使って行うことができます。

参照関数　acos, asin, atan, atan2, matherr, sin, sinh, tan, tanh

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        int     i;

        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("cos(%+4.1f) = %g¥n", x, cos(x));
        }
        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("cosh(%+4.1f) = %g¥n", x, cosh(x));
        }
}
```

cprintf

レベル2

書式

#include <conio. h>

int cprintf(format_string [, argument…]) ;
char ＊format_string ; /＊ 書式文字列へのポインタ ＊/

機能

format_string で指定した書式に従って出力を変換しコンソールに出力します。
format_string に続く引数は書式に従って変換され出力されます。
format_string の書式と機能は printf 関数と同じです。
詳細については、printf 関数を参照してください。
なお、cprintf は fprintf, printf 関数と異なり、改行 (¥n) を復帰改行 (¥m¥n) には変換しません。

戻り値

出力した文字数を返します。

参照関数

fprintf, printf, sprintf

プログラム例

```
#include        <conio.h>


main()
{
        cprintf("hello¥r¥n");
        cprintf("world¥r¥n");

        cprintf("hello¥n");
        cprintf("world¥n");

        cprintf("¥r¥n");

        cprintf("[%d]¥r¥n", 10);
        cprintf("[%s]¥r¥n", "string");
}
```

cputs

レベル 2

書　式

```
#include <conio.h>

int cputs(str) ;
char   *str ;  /*  出力文字列を指すポインタ  */
```

機　能

文字列 str をコンソールに出力します。
ただし文字列の末尾はヌル文字（'¥0'）でなくてはなりません。
また改行文字（'¥n'）を、復帰改行（¥r¥n）に変換しません。
正しく復帰改行したい場合は、¥r¥n として下さい。

戻り値

正常終了したときは、0 を返します。
エラーが発生した場合は、0 以外を返します。

参照関数

putch

プログラム例

```
#include        <conio.h>

main()
{
        cputs("hello¥r¥n");
        cputs("world¥r¥n");

        cputs("hello¥n");
        cputs("world¥n");
}
```

creat

レベル2

書　式

```
#include <io.h>
#include <stat.h>

int creat(pathname, pmode);
char  *pathname;  /*  作成するファイルのパス名へのポインタ  */
int pmode;  /*  アクセスモード  */
```

機　能

pathnameで指定したファイルが存在しない場合は、新しいファイルをpmodeで指定したアクセスモードで作成します。

pathnameで指定したファイルがすでに存在し、アクセスモードが書き込み可の場合は、既存のファイルの内容を削除して、書き込み用にオープンします。

pmodeはstat.hで定義されている定数S_IWRITE、S_IREADか、またはその2つを論理和した値を指定します。

pmodeの値とその内容についてはchmodを参照してください。

戻り値

ファイルが正しく作成された場合は、作成したファイルのファイルハンドルを返します。

正しく作成されなかった場合は−1を返し、errnoに以下の値がセットされます。

errnoの値	内　容
EACCES	指定したファイルはすでに存在し、読み出し専用のファイル、またはディレクトリ
EMFILE	ファイルハンドルがない
ENOENT	パス名がみつからない

参照関数　chmod, close, open

crt　　レベル3

書　式

```
#include <image.h>

void crt(screen_sw);
char screen_sw;  /* 切り替えスイッチ */
```

機　能

テレビ、ビデオ画面の切り替えを行います。
screen_sw に指定できる内容は次の通りです。

0 ：テレビ、ビデオの映像表示
1 ：コンピュータ画面の表示
2 ：テレビ、ビデオ映像のコントラストを下げてスーパーインポーズ表示
3 ：テレビ、ビデオ映像のコントラストを通常にしてスーパーインポーズ表示

本関数の実行前に、screen 関数により低解像度モード（512×512、または256×256）にしておく必要があります。
なお、crt（3）を実行すると、テロッパー機能によりコンピュータ画像やスーパーインポーズの画面をビデオに録画することができます。

戻り値

戻り値はありません。

cscanf

レベル2

書式

```
#include <conio.h>

int cscanf(format_string [, argument…]) ;
char *format_string ; /* 制御文字列 */
```

機能

コンソールからデータを読み込み、format_string の書式に従って変換し、後に続く argument の指すフィールドに代入します。
format_string の内容は scanf と同じです。
また引数 argument は、format_string の各変換指定に対応する型変数のポインタです。

戻り値

代入したフィールドの数を返します。
代入しなかったフィールドは含みません。
フィールドに1つも代入しなかった場合は0を返します。
読み込みの前に入力が尽きてしまった場合は EOF を返します。

参照関数

fscanf, scanf, sscanf

csrlin

レベル 3

書　　式

```
#include <basic0.h>

int csrlin( ) ;
```

機　　能　カーソルの現在位置のＹ座標を得ます。

戻 り 値　カーソルのＹ座標を返します。

参照関数　pos

ctime

レベル2

書　式

```
#include <time.h>

char *ctime(time);
time_t *time;
```

機　能

time_t 型の値の時刻 time を文字列に変換します。
得られる文字列の書式は asctime 関数と同じです。
ctime は内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。
また、前回の呼び出しで返された文字列は失われます。
それ以外にも、ctime は内部で localtime 関数を呼び出すので、それ以前に呼び出された、localtime 関数で返された構造体の内容も失われます。

戻り値

変換した文字列へのポインタを返します。

参照関数

asctime, ftime, gmtime, time

difftime

レベル2

書　式

```
#include <time.h>

double dfftime(time1, time2) ;
time_t time1 ;  /*  引かれる値  */
time_t time2 ;  /*  引く値  */
```

機　能

time1 から time2 を引いた時間を double で返します。
time_t 型の時間情報は、そのまま引き算する事ができないので、この difftime 関数を用いて時間差を求め、double で得ることができます。

戻り値

time1 から time2 を引いた時間を double で返します。

プログラム例

```
#include        <stdio.h>
#include        <time.h>


main()
{
        time_t  time1;
        time_t  time2;
        double  diff;
        int     i;

        time(&time2);
        for (i = 0; i < 0x2ffff; i++)
                ;
        time(&time1);
        diff = difftime(time1, time2);
        printf("%g秒¥n", diff);
}
```

div

レベル2

書　　式

#include <stdlib. h>

div_t div(number, denom) ;
int number ; /* 割られる値 */
int denom ; /* 割る値 */

機　　能

number を denom で割った商と余りを、div_t 型の構造体として返します。
div は符号付き整数の除算を行う場合に、処理系の違いによる結果の相違を解消するために用意されています。
div_t 構造体は、符号付き整数2つをメンバに持つ構造体で、stddef.h で以下のように型宣言されています。

```
typedef struct {
        int     quot ;
        int     rem ;
} div_t ;
```

戻 り 値

number を denom で割った商と余りを div_t 型の構造体として返します。
戻り値は、構造体の実体である事に注意して下さい。
構造体のメンバ quot には商が、メンバ rem には余りが返ります。

参照関数

ldiv

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        div_t   ans;
        int     num;
        int     denom;

        for (num = -10; num <= 10; num++) {
                for (denom = -10; denom <= 10; denom++) {
                        if (denom != 0) {
                                ans = div(num, denom);
                                printf("%d / %d = %d 余り %d¥n",
                                 num, denom, ans.quot, ans.rem);
                        }
                }
        }
}
```

dqsort, fqsort, lqsort, sqsort, tqsort

レベル 2

書式

```
#include <stdlib.h>

void dqsort(da, n) ;
double *da; /* ソートする配列へのポインタ (double型)  */
int n; /* 配列の要素数 */

void fqsort(fa, n) ;
float *fa; /* ソートする配列へのポインタ (float型)  */
int n; /* 配列の要素数 */

void lqsort(la, n) ;
long *la; /* ソートする配列へのポインタ (long型)  */
int n; /* 配列の要素数 */

void sqsort(sa, n) ;
short *sa; /* ソートする配列へのポインタ (short型)  */
int n; /* 配列の要素数 */

void tqsort(ta, n) ;
char **ta; /* ソートする配列へのポインタ (ポインタ型)  */
int n; /* 配列の要素数 */
```

機能

ソートを行います。
dqsort は double 型、fqsort は float 型、lqsort は long 型、sqsort は short 型、tqsort はポインタ型の配列をそれぞれソートします。

注：この他に文字列をソートする関数として、strsrt があります。
strsrt の機能は tqsort と同じですが、バブルソートアルゴリズムを用いているので、サイズの小さい配列に対して効果的です。

戻り値

戻り値はありません。

参照関数

strsrt

dskf

レベル 3

書　式

```
#include <basic.h>

int dskf(drive_no) ;
int drive_no; /* ドライブ番号 */
```

機　能

drive_no で指定したドライブのディスクの空き容量を調べます。
drive_no の指定できる内容は次の通りです。

0 ：カレントドライブ
1 ：ドライブA
2 ：ドライブB
⋮

戻り値

ディスクの空き容量をバイト数で返します。

dtasc　　レベル２

書　式

```
#include <io.h>

char *dtasc(date);
int date;  /*  文字列に変換したい日付情報  */
```

機　能

dateで指定された日付情報を文字列に変換します。
dateの内容は以下の通りです。

ビット 15～9：1980年からの相対年数(0～99)
ビット 8～5 ：月(1～12)
ビット 4～0 ：日(1～31)

dateの形式は、dtcnv関数の戻り値と同じ形式です。
得られる文字列は11文字で、フォーマットは以下のようになります。

86-09-29¥0

dtascは、内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。
また、前回の呼び出しで返された文字列は失われます。

戻り値

正常終了ならば、変換した文字列へのポインタを返します。
エラーならばNULLを返します。

参照関数

dtcnv

プログラム例

```
#include        <stdio.h>
#include        <io.h>
#include        <fcntl.h>
#include        <stat.h>

main()
{
        char    *p;
        int     fn;
        long    datetime;

        fn = open("¥¥CONFIG.SYS", O_RDWR);
        datetime = getft(fn);

        if ((p = dtasc(datetime >> 16)) != NULL);
                printf("日付: %s¥n", p);
        if ((p = tmasc(datetime & 0xffff)) != NULL);
                printf("時刻: %s¥n", p);
        close(fn);
}
```

dtcnv　　レベル2

書　式　#include <io. h>

int dtcnv(datestr) ;
char ＊datestr ;　/＊　変換したい日付文字列　＊/

機　能　datestrで指定された文字列を日付情報に変換して、その値を返します。
指定する文字列は11文字で、フォーマットは以下のように指定します。

86-09-29¥0

datestrのフォーマットは、dtasc関数の戻り値と同じフォーマットです。

戻り値　正常終了ならば、以下に示す形式で日付情報を返します。

ビット15〜9：1980年からの相対年数(0〜99)
ビット8〜5　：月(1〜12)
ビット4〜0　：日(1〜31)

エラーならば、負の数を返します。

参照関数　dtasc

プログラム例

```
#include        <stdio.h>
#include        <io.h>
#include        <fcntl.h>
#include        <stat.h>


main()
{
        int     fn;
        int     datetime;
        int     date;
        int     time;

        if ((date = dtcnv("90-03-14")) < 0)
                printf("エラー¥n");
        if ((time = tmcnv("12:30:00")) < 0)
                printf("エラー¥n");

        datetime = ((date << 16) | time);

        fn = open("¥¥CONFIG.SYS", O_RDWR);
        chgft(fn, datetime);
        close(fn);
}
```

dup, dup2

レベル2

書　式

```
#include <io.h>

int dup(handle);
int handle;  /*  ファイルハンドル  */

int dup2(handle1, handle2);
int handle1;  /*  ファイルハンドル  */
int handle2;  /*  新たに割りあてられるファイルハンドル  */
```

機　能

現在オープンされているファイルのファイルハンドルをコピーして、新しいファイルハンドルを作成します。

もとのファイルハンドルと新しいファイルハンドルは、同じファイルポインタを共有しますので、どちらのファイルハンドルを使ってもかまいません。

またファイルのアクセスモードは変化しません。

dup2 は、handle2 が handle1 と同じファイルを参照するようにします。

handle2 がオープンされているときは、まずクローズにした後に行います。

戻り値

dup の場合、新しいファイルハンドルを返します。

dup2 の場合は、正の数（0 も）で正常終了、−1 でエラーです。

正確には、複写先のファイルハンドラ（2 番目のパラメータの内容）を返します。

参照関数

close, creat, open

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>


main()
{
        int     fn1;
        int     fn2;

        if ((fn1 = open("dup.c", O_TEXT|O_RDWR)) == -1) {
                printf("open失敗¥n");
                exit(1);
        }
        printf("open %d¥n", fn1);
        if ((fn2 = dup(fn1)) == -1) {
                printf("dup()失敗¥n");
                exit(1);
        }
        printf("dup %d¥n", fn1);
        if ((fn2 = dup2(2, fn1)) == -1) {
                printf("dup2()失敗¥n");
                exit(1);
        }
        printf("dup2 %d¥n", fn2);
}
```

ecvt

レベル2

書　式

```
#include <stdlib.h>

char *ecvt(value, ndigits, decptr, signptr) ;
double value ; /*  変換すべき浮動小数点値  */
int ndigits ; /*  文字列の桁数  */
int *decptr ; /*  小数点位置を返すポインタ  */
int *signptr ; /*  符号を返すポインタ  */
```

機　能

浮動小数点値valueを、ndigitsで指定した桁数の文字列に変換します。
生成された文字列の末尾にヌル文字'¥0'をつけ加えます。
valueの桁数がndigitsの桁数より大きい場合は四捨五入され、小さい場合は文字列のうしろに0をつけます。
decptrは、小数点の位置を示す整数値で、0または負の数字の場合は小数点が左端にあることを示します。
signptrは、数値の符号を示す整数値で、0の場合は正の数、0以外の場合は負の数であることを示します。
文字列に変換できなかった時は、長さが0の文字列にします。
ecvtは、内部の静的な領域へのポインタを返しますので返されたポインタが指す領域を書き変えてはいけません。
また、前回の呼び出しで返された文字列は失われます。
なお、printfなどでも、%e、%Eの書式指定を使用すると、ecvtが呼び出されるので、それ以前にecvtで返された文字列は失われます。

戻り値

変換された文字列へのポインタを返します。

参照関数

atof, fcvt, gcvt

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define PI      3.1415926535897932385


main()
{
        int     dec;
        int     sign;
        char    *buf;
        int     n;

        for (n = 1; n <= 100; n++) {
                buf = ecvt(PI / n, 10, &dec, &sign);
                sub(buf, dec, sign);
        }
}


sub(char *buf, int dec, int sign)
{
        int     i;
        int     n;

        (sign == 0) ? putchar('+') : putchar('-');
        if (dec > 0)
                n = dec;
        else
                n = 1;
        for (i = 0; buf[i] != '¥0'; i++) {
                if (i == n)
                        putchar('.');
                putchar(buf[i]);
        }
        if (dec <= 0)
                printf("e%d", dec - 1);
        putchar('¥n');
}
```

eof

レベル2

書式

```
#include <io.h>

int eof(handle);
int handle;  /*  調べたいファイルのファイルハンドル  */
```

機能

handleで指定されるファイルが、エンドオブファイル（EOF）に達しているかどうかを調べます。

戻り値

現在の状態がエンドオブファイル（EOF）状態である場合は1を返し、そうでなければ0を返します。

エラーが起きたときは、−1を返しerrnoにEBADFがセットされます。

EBADFはファイルハンドルが適切でないことを示します。

参照関数

clearerr, feof, ferror

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>

#define SIZE    16


main(int argc, char *argv[])
{
        char    buf[SIZE];
        char    *file;
        int     fn;
        int     n;

        file = (argc > 1) ? argv[1] : "file";

        if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
                printf("can't open %s¥n", file);
                exit(1);
        }
        while (!eof(fn)) {
                n = read(fn, buf, SIZE);
                printf("%d bytes readed.¥n", n);
        }
        close(fn);
        exit(0);
}
```

execl, execle, execlp, execlpe, execv, execve, execvp, execvpe　レベル 2

書　　式

```
#include <process.h>

int execl(pathname, arg0, arg1…argn, NULL) ;
int execle(pathname, arg0, arg1…argn, NULL, envp) ;
int execlp(pathname, arg0, arg1…argn, NULL) ;
int execlpe(pathname, arg0, arg1…argn, NULL, envp) ;
int execv(pathname, argv) ;
int execve(pathname, argv, envp) ;
int execvpe(pathname, argv, envp) ;
int execvp(pathname, argv) ;
char *pathname; /* 実行ファイルのパス名へのポインタ */
char *arg0, *arg1, …, *argn; /* 引数へのポインタリスト */
char *argv []; /* 引数へのポインタ配列 */
char *envp []; /* 環境へのポインタ配列 */
```

機　　能

新しい子プロセスを起動して制御を移します。
メモリ内では新しい子プロセスは、呼び出したプロセスの次にロードされます。
このため、ロードができるだけのメモリ領域が必要です。
pathname は子プロセスのあるファイル名のパス名を指定します。
パス名に拡張子があれば、その拡張子で探しますが、指定がなければ、"X"ファイルを探します。
パス名がピリオドで終わっている場合は、拡張子のないファイルを探します。
引数はそのまま、新しいプロセスにわたされます。
パス名と引数あわせて 255 バイトまでで指定します。
文字列末のヌル文字もこのバイト数に含まれますので注意してください。
引数として、最小限、arg0 か argv [0] はわたさなくてはなりません。
execl、execle、execlp および execlpe（関数に"l"のつくもの）は、引数をポインタのリストでわたします。
arg0 は使用されません。
execv、execvp、execve および execvpe（関数に"v"のつくもの）は、引数をポインタの配列でわたします。
argv [0] は使用されません。
execle、execlpe、execve および execvpe（関数"e"のつくもの）は envp 引数で環境設定のリストをわたし、環境を指定します。

envpは文字列を指すポインタの配列で、各文字列は、

NAME = value

とします。

ここで、NAMEは環境変数名で、valueはその値です。

envpがNULLの場合、親プロセスの環境を引き継ぎます。

execlp、execlpe、execvpおよびexecvpe（関数に“p”のつくもの）は、path環境変数でディレクトリの指定ができます。

戻り値

正常終了した場合は正の数（0も）を返します。

これは起動したプロセスの終了コードです。

エラーが発生した場合は−1を返します。

その場合errnoに下記の値がセットされます。

errnoの値	内　容
E2BIG	引数リストが256バイトを超えている
EMFILE	オープンファイルの数が多すぎる
ENOENT	ファイルまたはパス名が見つからない
ENOEXEC	指定のファイルが実行できない
ENOMEM	子プロセスを実行するためのメモリが不足している

参照関数

abort, exit, _exit, spawnl, spawnle, spawnlp, spawnv, spawnve, spawnvp, system

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <process.h>


main(int argc, char *argv[])
{
        char    *args[4];
        char    *envs[3];
        char    *pathname;

        pathname = argv[1];
        args[0] = argv[0];
        args[1] = argv[2];
        args[2] = argv[3];
        args[3] = NULL;
        envs[0] = "path=A:¥¥bin¥¥";
        envs[1] = "include=¥¥";
        envs[2] = NULL;

        if (execl(pathname, args[0], args[1], args[2], NULL) == -1)
                printf("失敗しました。¥n");
        if (execv(pathname, args) == -1)
                printf("失敗しました。¥n");
        if (execvs(pathname, args, envs) == -1)
                printf("失敗しました。¥n");
        if (execvp(pathname, args) == -1)
                printf("失敗しました。¥n");
}
```

except

レベル2

書式

```
#include <math.h>

double except(type, name, arg1, arg2, retval) ;
int type ; /* 数値演算エラーのタイプ */
char *name ; /* 例外条件が発生した関数名へのポインタ */
double arg1 ; /* 上記関数呼び出し時の引数 */
double arg2 ; /* 上記関数呼び出し時の引数 */
double retval ; /* デフォルトの戻り値 */
```

機能

標準数学ライブラリ関数は、例外条件が発生したときにはmatherr関数を呼び出すように作られていますが、matherr関数の呼び出しを簡略化するための関数として、exceptが用意されています。

exceptは、パラメータをセットしてmatherr関数を呼び出し、matherr関数の返す値に従って適当な値を返します。

typeには、数値演算エラーのタイプを指定します。

詳細については、matherr関数を参照してください。

nameには、例外条件が発生した関数名を格納する文字列（ヌル文字で終了）へのポインタを指定します。

arg1、arg2は数値演算関数に指定された引数を保持しますが、引数が1つであれば、arg1に格納され、arg2は使用されません。

retvalはエラーが発生した数値演算関数のデフォルトの戻り値です。

type, name, arg1, arg2, retvalは構造体exceptionの各メンバに格納され、その構造体へのポインタがmatherr関数に渡されます。

そして、matherr関数からの戻り値が、例外回復処理を行ったことを示す値だったならば、構造体exceptionのメンバretvalから値をとり出してそれを返します。

例外回復処理を行わなかったことを示す値だったならば、デフォルトの戻り値であるretvalを返します。

戻り値

matherrを呼び出して例外処理を行い、その結果を返します。

参照関数

matherr

exit, _exit

レベル2

書　式

```
#include <stdlib.h>

void exit(status) ;
void _exit(status) ;
int status;  /*実行終了時のステータス  */
```

機　能

実行を終了させます。
exitは、プログラムの終了前にatexit関数で登録されていた関数を、登録した順と逆の順に呼び出します。
そしてすべてのバッファをフラッシュし、現在オープンしているすべてのファイルをクローズします。
_exitは何もせず、実行の終了のみを行います。
通常は、statusには、正常に終了する場合は定数として宣言されているEXIT_SUCCESSを、そうでない場合には定数として宣言されているEXIT_FAILUREをセットします。

戻り値

戻り値はありません。

参照関数

abort, atexit

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "r")) == NULL) {
                printf("can't open %s¥n", file);
                exit(EXIT_FAILURE);
        }
        (void)fclose(fp);
        printf("normal end¥n");
        exit(EXIT_SUCCESS);
}
```

exp

レベル２

書　式

include <math. h>

double exp(x) ;
double x ;　/*　浮動小数点値　*/

機　能

自然対数の底 e の x 乗を返します。

戻り値

e^xを返します。
オーバーフローが発生した場合、ERANGE を errno にセットします。
またエラー処理は、matherr 関数を使って行うことができます。

参照関数

log

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        int     i;

        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("exp(%+4.1f) = %g¥n", x, exp(x));
        }
}
```

fabs

レベル2

書　式

#include <math. h>

double fabs(x) ;
double x ; /* 浮動小数点値 */

機　能

浮動小数点値 x の絶対値を計算します。

戻り値

引数 x の絶対値を返します。

参照関数

abs, cabs, labs

プログラム例

```
#include        <stdio.h>
#include        <math.h>

main()
{
        double  x;
        int     i;

        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("fabs(%+4.1f) = %g¥n", x, fabs(x));
        }
}
```

fclose, fcloseall　レベル 2

書　式

```
#include <stdio.h>

int fclose(stream) ;
FILE  *stream ;  /*  FILE構造体へのポインタ  */
int fcloseall( ) ;
```

機　能

fcloseall は stdin、stdout、stderr、stdaux、stdprn 以外のオープン中のストリームをクローズします。

fclose は指定したストリームだけをクローズします。

ストリームに関連するすべてのバッファはフラッシュされ、またシステムが割りつけたバッファも解放されます。

ただし、setbuf を使って割りつけたバッファは解放されません。

stdin, stdout に対して fclose を行うと、入出力のリダイレクトが無効になりますので注意してください。

戻り値

fclose ではストリームを正常にクローズした場合は 0 を返し、fcloseall ではクローズしたストリームの数を返します。

正常にクローズできなかった場合は EOF を返します。

参照関数

close, fdopen, fflush, fopen, freopen

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     n;

        file = (argc > 1) ? argv[1] : "file";

        if ((fp = fopen(file, "r")) == NULL) {
                printf("%sをopenできません。¥n", file);
                exit(1);
        }
        printf("%sをopenしました。¥n", file);

        if (fclose(fp) == EOF) {
                printf("%sをcloseできません。¥n", file);
                exit(1);
        }
        printf("%sをcloseしました。¥n", file);

        n = fcloseall();
        printf("%d個のファイルをcloseしました。¥n", n);
        exit(0);
}
```

fcvt　レベル2

書　式　# include <stdlib. h>

```
char  *fcvt(value, ndec, decptr, signptr) ;
doule value ; /*  変換すべき浮動小数点値  */
int ndec ; /*  小数点以下の有効桁数  */
int  *decptr ; /*  小数点位置を返すポインタ  */
int  *signptr ; /*  符号を返すポインタ  */
```

機　能　浮動小数点値 value を文字列に変換し、末尾にヌル文字'¥0'をつけ加えます。
ndec は小数点以下の桁数で、value の小数点以下の桁数が ndec よりも大きい場合は四捨五入されます。小さいときは 0 をつけ加えます。
数字以外は変換されません。
decptr は左端から小数点の位置を示す整理値です。
0 または負ならば、小数点位置は左端となります。
signptr は value の符号で、0 の場合は正、それ以外の場合は負を示します。
文字列に変換できなかった時は、長さが 0 の文字列にします。
fcvt は、内部の静的な領域へのポインタを返しますので、返されたポインタが指す領域を書き変えてはいけません。
また、前回の呼び出しで返された文字列は失われます。
なお、print などでも%f、%F の書式指定を使用すると fcvt が呼び出されるので、それ以前に fcvt で返された文字列は失われます。

戻り値　変換された文字列へのポインタを返します。

参照関数　atof, ecvt, gcvt

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define PI      3.14159265358979323846


main()
{
        int     dec;
        int     sign;
        char    *buf;
        int     n;

        for (n = 1; n <= 100; n++) {
                buf = fcvt(PI / n, 10, &dec, &sign);
                sub(buf, dec, sign);
        }
}


sub(char *buf, int dec, int sign)
{
        int     i;
        int     n;

        (sign == 0) ? putchar('+') : putchar('-');
        if (dec > 0)
                n = dec;
        else
                n = 1;
        for (i = 0; buf[i] != '¥0'; i++) {
                if (i == n)
                        putchar('.');
                putchar(buf[i]);
        }
        if (dec <= 0)
                printf("e%d", dec - 1);
        putchar('¥n');
}
```

fdelete

レベル3

書　式

```
#include <basic.h>

int fdelete(f_name);
char  *f_name;  /*  ファイル名格納領域へのポインタ  */
```

機　能　f_nameが指すファイルを削除します。

戻り値　成功すると0を、失敗すると−1を返します。

fdopen

レベル 2

書　式

#include <stdio. h>

FILE　*fdopen(handle, type)；
int handle；　/*　該当ファイルのファイルハンドル　*/
char　*type；　/*　アクセスモード格納領域へのポインタ　*/

機　能

open などでオープンした、handle で指定されたファイルをストリームに対応させます。
type はストリームのアクセスモードを指定する文字列で、詳しくは fopen 関数を参照してください。

戻り値

ストリームのポインタを返します。
エラーならば NULL を返します。

参照関数

dup, dup2, fclose, fcloseall, fopen, freopen, open

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <stat.h>
#include        <io.h>


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     fn;

        file = (argc > 1) ? argv[1] : "file";

        if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
                printf("can't open %s¥n", file);
                exit(1);
        }
        if ((fp = fdopen(fn, "r")) == NULL) {
                printf("%sをopenできません。¥n", file);
                exit(1);
        }
        (void)fclose(fp);
        printf("normal end.¥n");
        exit(0);
}
```

feof

レベル２

書　式

```
#include <stdio.h>

int feof(stream);
FILE *stream; /* FILE構造体へのポインタ */
```

機　能

指定されたストリームがエンドオブファイル（EOF）になったかどうかを調べます。

一度エンドオブファイルになると、ストリームがクローズするか rewind 関数を実行するまで、エンドオブファイル状態のままになります。

戻り値

現在の位置がエンドオブファイルでない場合は0を返します。

エンドオブファイルの場合は、0以外の値を返します。

参照関数

clearerr, eof, ferror

注：feof はマクロ定義と関数の両方があります。

マクロ定義で使いたい場合は、stdio. h をインクルードする前に次の1行を追加してください。

```
#define MACRO
```

ferror

レベル2

書　式

#include <stdio. h>

int ferror(stream) ;
FILE　*stream ;　/*　FILE構造体へのポインタ　*/

機　能

指定したストリームの入出力時にエラーが発生しているかどうかを判定します。エラーが発生した場合は、ストリームがクローズされるか、またはrewindかclearerr関数が呼ばれるまで、エラー状態のままになっています。

戻り値

エラーがない場合は0を、エラーがある場合は0以外の値を返します。

参照関数

clearerr, eof, feof, fopen

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     c;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "r")) == NULL) {
                perror(file);
                exit(1);
        }
        while ((c = getc(fp)) != EOF) {
                if (ferror(fp)) {
                        perror(file);
                        clearerr(fp);
                        break;
                }
        }
        (void)fclose(fp);
        exit(0);
}
```

注：ferrorはマクロ定義と関数版があります。
　マクロ定義で使いたい場合は、stdio. hをインクルードする前に次の1行を追加してください。

#define MACRO

fflush

レベル2

書式

```
#include <stdio.h>

int fflush(stream) ;
FILE  *stream ;  /*  FILE構造体へのポインタ  */
```

機能

出力ストリームと関連するバッファ内のデータが、ファイルに書かれていないならば、それをファイルに書き込みます。
これにより、バッファが空になります。
書き込み後もストリームはオープンしたままです。
また、ファイルポインタも変化しません。
入力ストリームと関連するバッファ内にデータがあれば、それは捨てられバッファが空になります。
データを捨てるだけでストリームはオープンしたままです。
また、ファイルポインタも変化しません。
なお、stdin は、通常ではバッファを使っていませんが、Human68k 内部でバッファを持っていますので、fflush ではこれをフラッシュできません。

ファイルポインタとは、ファイル中のどの位置で読み書きが行われるかを保持している指標のことです。
バッファを使っていないストリームに対して、この関数を実行しても影響はありません。
stream に NULL を指定すると、すべての入出力ストリームに対して処理が行われます。
なお、クローズやフラッシュを行わずに exit 以外でプログラムを終了すると、最後にストリームに出力されたデータがバッファに残ったまま捨てられてしまいますので、必ず fflush してください。

戻り値

バッファ内のデータが正常に書き込めたときには0を返します。
指定のストリームがバッファを使っていないか、または読み出し専用でオープンしている場合にも0を返します。
エラーが起きたときは EOF を返します。
stream に NULL を指定したときは、常に0を返します。

参照関数

fclose, flushall, setbuf

注：XC では fflush で NULL が指定できますが、他のCコンパイラではできない可能性があるので注意してください。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     c;

        file = (argc > 1) ? argv[1] : "outfile";
        if ((fp = fopen(file, "w")) == NULL) {
                perror(file);
                exit(1);
        }
        while ((c = getchar()) != EOF) {
                putc(c, fp);
                if (ferror(fp)) {
                        perror(file);
                        clearerr(fp);
                        break;
                }
        }
        if (fflush(fp) == EOF) {
                perror(file);
                clearerr(fp);
        }
        (void)fclose(fp);
        exit(0);
}
```

注：バッファがいっぱいになったとき、ストリームがクローズされたとき、ストリームをクローズしないでプログラムがexitによって終了したときには、自動的にフラッシュが行われます。

ただし、プログラム終了時に自動的にフラッシュしようとして発生したエラーは無視されてしまいますので、この場合には、書き込もうとしたデータが失われてしまう可能性があります。

fgetc, fgetchar　レベル 2

書　式　# include <stdio. h>

int fgetc(stream) ;
FILE　*stream ;　/*　FILE構造体へのポインタ　*/

int fgetchar ( ) ;

機　能　fgetcはstreamで指定するストリームから1文字読み込みます。
fgetcharはfgetc (stdin) と同じです。

戻り値　読み込んだ文字を返します。
エラー、またはエンドオブファイルのときはEOFを返します。
EOFが返された時にエラーかエンドオブファイルどちらが発生したかを判定するためには、feofまたはferror関数を使用してください。

参照関数　fputc, fputchar, getc, getchar

プログラム例

```
#include        <stdio.h>

main(int argc, char *argv[])
{
        FILE    *fp;
        int     c;

        if (argc == 1) {
                while ((c = fgetchar()) != EOF)
                        putchar(c);
        }
        else {
                if ((fp = fopen(argv[1], "r")) == NULL) {
                        perror(argv[1]);
                        exit(1);
                }
                while ((c = fgetc(fp)) != EOF)
                        putchar(c);
                (void)fclose(fp);
        }
        exit(0);
}
```

fgetpos

レベル2

書　式

```
#include <stdio.h>

int fgetpos(stream, pos) ;
FILE   *stream ;  /*  FILE構造体へのポインタ  */
fpos_t   *pos ;  /*  求めたアクセス位置を格納するポインタ  */
```

機　能

streamが指すストリームの現在のアクセス位置を求め、posが指すポインタへ格納します。
異なる処理系の間で移植などを行う場合、処理系の違いなどにより、ストリームのアクセス位置を表す情報に違いがあるため、fseekやftell関数では十分なアクセスが行えない場合があります。
fgetposは、そのような問題を解消し、移植しやすいプログラムを記述する為に用意されています。
XCではfpos_t型は、stddef.hで以下のように型宣言されています。

```
typedef long    fpos_t ;
```

戻り値

成功すると0を、失敗すると0以外を返します。

参照関数

fsetpos, ftell, fseek

fgets　　レベル 2

書　式　# include <stdio. h>

char　＊fgets(string, n, stream)；
char　＊string；　/＊　読み込んだ文字列を格納する領域へのポインタ　＊/
int n；　/＊　読み込む文字数　＊/
FILE　＊stream；　/＊　FILE構造体へのポインタ　＊/

機　能　streamで指定するストリームから改行文字'¥n'が現われるか、ストリームがエンドオブファイルになるか、または読み込んだ文字数がn−1になるまで文字列を読み込みます。
読み込んだ文字列の末尾にはヌル文字'¥0'が付加され、stringが指す領域に格納されます。
nが1のときは、読み込んだ文字列は空文字列になります。

戻り値　読み込んだ文字列へのポインタを返します。
エラーが発生した場合またはエンドオブファイルを読み込んだ場合はNULLを返します。
どちらによるNULLかは、ferrorまたはfeof関数を使用して調べることができます。

参照関数　fputs, gets, puts

プログラム例

```
#include        <stdio.h>

#define SIZE    1024


main(int argc, char *argv[])
{
        char    buf[SIZE];
        char    *file;
        FILE    *fp;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "r")) == NULL) {
                perror(file);
                exit(1);
        }
        while (fgets(buf, SIZE, fp) != NULL)
                fputs(buf, stdout);
        (void)fclose(fp);
        exit(0);
}
```

filelength

レベル2

書　式　#include <io. h>

long filelength(handle) ;
int handle ;　/*　調べたいファイルのファイルハンドル　*/

機　能　指定したhandleのファイルの大きさを調べます。

戻り値　ファイルの大きさをバイト数で返します。
エラーが起きたときは−1を返し、errnoにEBADFをセットします。

参照関数　chsize, fileno

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <io.h>


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     len;

        while (--argc) {
                file = *++argv;
                if ((fp = fopen(file, "r")) == NULL) {
                        perror(file);
                        exit(1);
                }
                len = filelength(fileno(fp));
                printf("name = %s, length = %d¥n", file, len);
                (void)fclose(fp);
        }
}
```

fileno

レベル 2

書　　式　#include <stdio. h>

int fileno(stream) ;
FILE　*stream ;　/*　FILE構造体へのポインタ　*/

機　　能　指定したストリームと関連しているファイルハンドルを調べます。

戻 り 値　ファイルハンドルを返します。
複数のファイルハンドルがストリームと関連している場合は、ストリームが最初にオープンしたときに割りつけたファイルハンドルを返します。
ストリームがオープン済みのファイルを持たない場合の戻り値は不定です。

参照関数　fdopen, filelength, fopen, freopen

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <io.h>

main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     len;

        while (--argc) {
                file = *++argv;
                if ((fp = fopen(file, "r")) == NULL) {
                        perror(file);
                        exit(1);
                }
                len = filelength(fileno(fp));
                printf("name = %s, length = %d¥n", file, len);
                (void)fclose(fp);
        }
}
```

注：fileno はマクロ定義と関数版とがあります。
マクロ定義で使いたい場合は、stdio. h をインクルードする前に次の1行を追加してください。

#define MACRO

files

レベル 2

書　式

```
#include <io.h>

char  *files(buff, atr, datetime, len, pathname, src_atr) ;
char  *buff ;  /*  検索用バッファへのポインタ  */
char  *atr ;  /*  見つけたファイルの属性を格納する領域へのポインタ  */
int  *datetime ;  /*  見つけたファイルの作成日時を格納する領域へのポイ
                      ンタ  */
size_t  *len ;  /*  見つけたファイルのバイト数を格納する領域へのポインタ  */
char  *pathname ;  /*  検索するファイルのパス名へのポインタ  */
char  src_atr ;  /*  検索する属性  */
```

機　能

pathname と src_atr で指定したファイルを検索し、見つけたファイルの情報を atr, datesime, len で指定した領域に格納し、戻り値として見つけたファイルのファイル名へのポインタを返します。

buff は，ファイルを検索するときにこの関数が使用するバッファへのポインタです。

バッファの容量は 150 バイト必要です。

atr は，見つけたファイルの属性を格納する領域へのポインタです。

atr の指す領域のサイズは 1 バイト以上必要です。

* atr に格納される属性は、各ビットが 1 の場合に以下の意味を表しています。

ビット 0：読み込み専用
ビット 1：隠しファイル
ビット 2：システムファイル
ビット 3：ボリューム名
ビット 4：ディレクトリ
ビット 5：通常ファイル

datetime は、見つけたファイルの作成日時を格納する領域へのポインタです。

datetime の指す領域のサイズは 4 バイト以上必要です。

また、* datetime に格納される作成日時の形式は以下の通りです。

ビット 31～25　：1980 年からの相対年数(0～99)
ビット 24～21　：月(1～12)
ビット 20～16　：日(1～31)
ビット 15～11　：時(0～23)
ビット 10～5　 ：分(0～59)

ビット 4〜0　：秒/2(0〜29)

len は、見つけたファイルのバイト数を格納する領域へのポインタです。
len の指す領域のサイズは 4 バイト以上必要です。
pathname には、検索したいファイルのパス名へのポインタを指定します。
パス名にはワイルドカードが使用できるので、"A：¥*.*" という設定も可能です。
src_atr は検索したいファイルの属性を指定します。
src_atr の形式は以下の通りです。

0x01 ：読み込み専用
0x02 ：隠しファイル
0x04 ：システムファイル
0x08 ：ボリューム名
0x10 ：ディレクトリ
0x20 ：通常ファイル

この src_atr で示される属性と、ファイルの属性との論理積（AND）を取った値が 0 でなければ、検索の対象となります。
したがって、すべての属性を検索の対象にする場合は、0x3F を指定します。
また、読み込み専用の通常ファイルを検索の対象にしようとして、0x21 を指定しても、すべての通常ファイルと、すべての読み込み専用のボリューム名、ディレクトリ、通常ファイルが対象となってしまうことに注意してください。
files 関数を実行した後で、nfiles 関数をエラーが返されるまで実行することで、pathname と src_atr で指定したファイルすべてを検索することができます。

戻り値

正常終了ならば、見つけたファイルのファイル名へのポインタを返します。
なお、このファイル名は buff で指定した検索用バッファに作成されるため、同一の検索用バッファを指定して files 関数や nfiles 関数を呼び出すと以前の内容が失われることに注意してください。
エラーならば NULL を返します。

参照関数

chgfa, chgft, nfiles

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <io.h>

#define A_RDO           0x01    /* 読み込み専用 */
#define A_HDN           0x02    /* 隠しファイル */
#define A_SYS           0x04    /* システムファイル */
#define A_VOL           0x08    /* Volumeラベル */
#define A_DIR           0x10    /* ディレクトリ */
#define A_FIL           0x20    /* 通常ファイル */

static char     *pattern = "¥¥usr¥¥include¥¥*.h";
static char     buf[150];
static char     atr;
static long     dt;
static size_t   len;


main()
{
        char    *name;

        name = files(buf, &atr, &dt, &len, pattern, A_FIL);
        if (name != NULL)
                printf("%s¥n", name);
        else
                exit(1);

        while ((name = nfiles(buf, &atr, &dt, &len)) != NULL)
                printf("%s¥n", name);

        exit(0);
}
```

fill

レベル3

書式 #include <graph. h>

void fill(x1, y1, x2, y2, p) ;
int x1 ; /* 始点X座標 */
int y1 ; /* 始点Y座標 */
int x2 ; /* 終点X座標 */
int y2 ; /* 終点Y座標 */
int p ; /* パレットコード */

機能 グラフィック画面に塗りつぶした長方形を描画します。

x1には始点X座標を指定します（−32768-32767）。
y1には始点Y座標を指定します（−32768〜32767）。
x2には終点X座標を指定します（−32768〜32767）。
y2には終点Y座標を指定します（−32768〜32767）。
pにはパレットコードを指定します（0〜65535）。
画面モードにより最大値が異なります。

戻り値 戻り値はありません。

プログラム例

```
#include        <stdlib.h>
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i;
        int     x1, x2, y1, y2;
        int     pc;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        for (i = 0; i < 500; i++) {
                x1 = rand() % 768;
                y1 = rand() % 512;
                x2 = rand() % 768;
                y2 = rand() % 512;
                pc = rand() % 16;
                fill(x1, y1, x2, y2, pc);
        }
        b_exit(0);
}
```

fix

レベル3

書　式

```
#include <basic.h>

double fix(f);
double f;  /*  浮動小数点値  */
```

機　能

浮動小数点値 f の整数部を求めます。

戻り値

浮動小数点値の整数部を返します。

floor

レベル 2

書　式　#include <math.h>

double floor(x) ;
double x ;

機　能　x 以下で最大の整数を求めます。

戻り値　double としての整数を返します。

参照関数　ceil, modf

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        int     i;

        for (i = -20; i <= 20; i++) {
                x = (double)i / 10.0;
                printf("floor(%+4.1f) = %g¥n", x, floor(x));
        }
}
```

flushall

レベル２

書　式

#include <stdio.h>

int flushall( ) ;

機　能

オープンされているすべてのストリームに対して fflush を行います。
ストリームはクローズされません。
また、ファイルポインタも変化しません。
詳細については、fflush 関数を参照してください。

戻り値

オープンされているストリームの総数を返します。
エラーを表す戻り値は返しません。

参照関数

fflush

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        int     n;

        n = flushall();
        printf("%d個のstreamがopenされています。¥n", n);
        exit(0);
}
```

注：バッファがいっぱいになったときや、ストリームがクローズされたとき、ストリームをクローズしないでプログラムが exit によって終了したときには、自動的にフラッシュが行われます。
ただし、プログラム終了時に自動的にフラッシュしようとして発生したエラーは無視されてしまいますので、この場合には、書き込もうとしたデータが失われてしまう可能性があります。

fmod

レベル 2

書　　式

#include <math. h>

double fmod(x, y) ;
double x ;
double y ;

機　　能

x/y の剰余（浮動小数点値）を計算します。
x ＝ i×y＋f となる浮動小数点値 f を返します。
ここで、i は整数、f は x と同じ符号の値です。

戻 り 値

計算結果を返します。
y が 0 か x/y がオーバーフローになる場合は errno に EDOM がセットされ、0 を返します。
またエラー処理は、matherr 関数を使って行うことができます。

参照関数

ceil, fabs, floor

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        double  y;
        int     i;
        int     j;

        for (i = -32; i <= 32; i++) {
                for (j = -32; j <= 32; j++) {
                        x = (double)i / 10.0;
                        y = (double)j / 10.0;
                        printf("fmod(%+4.1f, %+4.1f) = %g¥n",
                         x, y, fmod(x, y));
                }
        }
}
```

fmode

レベル2

書　式

```
#include <stdio.h>

void fmode(fp, mode) ;
FILE  * fp;  /*  FILE構造体へのポインタ  */
int mode;  /*  ファイル変換モード  */
```

機　能

fpが指すストリームのファイル変換モード（テキストかバイナリ）を変更します。

modeは0か1を指定します。

0のときは変換モード（テキスト）になり、1のときは無変換モード（バイナリ）になります。

テキストモードの場合、入力において復帰改行'¥r¥n'を改行'¥n'に、出力において改行'¥n'を復帰改行'¥r¥n'に変換します。

ファイル変換モードを変更する場合は、fopenかfflush関数を行った直後にしてください。

戻り値

戻り値はありません。

fopen

レベル2

書　式

```
#include <stdio.h>

FILE  *fopen(pathname, type);
char  *pathname;  /*  ファイルのパス名へのポインタ  */
char  *type;  /*  アクセスモード格納領域へのポインタ  */
```

機　能

pathnameで指定したファイルをオープンします。
typeはファイルアクセスモードを指定する文字列で、次の通りです。

typeの値	内　容
"r"	ファイルを読み出す（ファイルは存在していること）
"w"	書き込みのためにファイルを新規作成する（同じファイルがあると前の内容は失われる）
"a"	ファイルの最後に追加書き込みをする（ファイルがないと新規作成）
"r+"	更新（読み込みと書き込み）する（ファイルは存在していること）
"w+"	更新（読み込みと書き込み）のためにファイルを新規作成する（同じファイルがあると前の内容は失われる）
"a+"	読み込みと追加書き込みする（ファイルがないと新規作成）

typeの指定は、ファイルの作成時のアクセスモードと矛盾してはいけません。
"a"または"a+"でファイルをオープンした場合には、ファイルポインタはファイルの最後へ移動し、オープン直後の書き込み操作はファイルの最後から行われます。
"a"、"a+"以外でオープンした場合には、ファイルポインタはファイルの先頭に位置します。
"r+"、"w+"、"a+"が指定された場合には、読み込みと書き込みの両方が可能となります（これをファイルの更新のためにオープンするといいます）。
しかし、読み込みから書き込みへ、またはその逆に移る場合には、fflush、fseekまたはrewind関数によるバッファの書き出し操作が必要です。
ファイルポインタはfseek関数により移動することができます。

なお、ファイルポインタとは、ファイル中のどの位置で読み書きが行われるかを保持している指標のことです。
さらに改行の変換モードを指定するために下記の文字を使用します。

typeの値	内　　容
“t”	テキストモードでオープンする 入力時には、復帰改行‘¥r¥n’を改行‘¥n’に、出力時には改行‘¥n’を復帰改行‘¥r¥n’に変換する
“b”	バイナリモードでオープンする

“t”または“b”がtype文字列にない場合、変換モードはデフォルトのモードを指示するグローバル変数_fmodeによって定義されます。

戻り値　オープンしたストリームのポインタを返します。
エラーが起きたときはNULLを返します。

参照関数　fclose, fcloseall, fdopen, ferror, fileno, freopen, open, setmode

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     n;

        file = (argc > 1) ? argv[1] : "file";

        if ((fp = fopen(file, "r")) == NULL) {
                printf("%sをopenできません。¥n", file);
                exit(1);
        }
        printf("%sをopenしました。¥n", file);

        (void)fclose(fp);
        printf("%sをcloseしました。¥n", file);

        exit(0);
}
```

fprintf

レベル 2

書式	#include <stdio. h> int fprintf(stream, format_string〔, argument. . . 〕); FILE　*stream;　/*　FILE構造体へのポインタ　*/ char　*format_string;　/*　書式文字列へのポインタ　*/
機能	format_stringで指定した書式に従って出力を変換しstreamで指定されたストリームに出力します。 format_stringに続く引数は書式に従って変換され出力されます。 format_stringの書式と機能はprintf関数と同じです。 詳細については、printf関数を参照してください。
戻り値	出力した文字数を返します。
参照関数	cprintf, fscanf, printf, sprintf

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        char    *file = "outfile";
        FILE    *fp;

        if ((fp = fopen(file, "w")) == NULL) {
                perror(file);
                fp = stderr;
        }
        fprintf(fp, "string->[%s]¥n", "hello");
        fprintf(fp, "decimal->[%d]¥n", 123);
        fprintf(fp, "floating->[%g]¥n", 3.14);
        (void)fclose(fp);
        exit(0);
}
```

fputc, fputchar　レベル2

書　式

#include <stdio. h>

int fputc(c, stream) ;
int c ; /*　出力する文字　*/
FILE　*stream ; /*　FILE構造体へのポインタ　*/

int fputchar(c) ;
int c ; /*　出力する文字　*/

機　能

fputc は、stream で指定するストリームに1文字出力します。
fputchar は、fputc(c, stdout) と同じになります。

戻り値

出力した文字を返します。
エラーが発生した場合は、EOF を返します。

参照関数

fgetc, fgetchar, putc, putchar

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        int     c;

        while ((c = getchar()) != EOF)
                fputc(c, stdout);
        exit(0);
}
```

fputs
レベル 2

書式

#include <stdio.h>

int fputs(string, stream) ;
char *string; /* 出力文字列へのポインタ */
FILE *stream; /* FILE 構造体へのポインタ */

機能

stream で指定されたストリームに文字列 string を出力します。
文字列の末尾のヌル文字'¥0'は出力されません。

戻り値

正常終了したときは 0 を返します。
エラーが発生した場合は、0 以外を返します。

参照関数

fgets, gets, puts, fputs

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define SIZE    1024


main(int argc, char *argv[])
{
        char    buf[SIZE];
        char    *file;
        FILE    *fp;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "r")) == NULL) {
                perror(file);
                exit(1);
        }
        while (fgets(buf, SIZE, fp) != NULL)
                fputs(buf, stdout);
        (void)fclose(fp);
        exit(0);
}
```

fread

レベル２

書　　式

```
#include <stdio.h>

size_t fread(buffer, size, count, stream) ;
void   *buffer ;  /*  読み込んだデータを格納する領域を指すポインタ  */
size_t size ;  /*  データ項目のサイズ  */
size_t count ;  /*  読み込みデータ項目数  */
FILE   *stream ;  /*  FILE構造体へのポインタ  */
```

機　　能

streamで指定されたストリームからsizeバイトの長さのデータ項目をcountの数だけ読み込み、bufferが指す領域に格納します。
ファイルポインタは、読んだバイト数だけ進みます。
ストリームがテキストモードでオープンされている場合は、復帰改行'¥r¥n'は改行'¥n'に変換されます。
この変換は、ファイルポインタや戻り値に影響を与えません。

戻 り 値

実際に読み込んだデータ項目の数を返します。
エラーが発生した場合や、指定したデータの項目数まで達しないうちにエンドオブファイル（EOF）になった場合には、countより少ない数が返されます。

参照関数

fwrite, read

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define SIZE    1024


main(int argc, char *argv[])
{
        char    buf[SIZE];
        char    *file;
        FILE    *fp;
        int     n;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "rb")) == NULL) {
                perror(file);
                exit(1);
        }
        while ((n = fread(buf, sizeof(char), SIZE, fp)) > 0)
                fwrite(buf, sizeof(char), n, stdout);
        (void)fclose(fp);
        exit(0);
}
```

free　レベル2

書　式　#include <stdlib. h>

int free(ptr) ;
void　＊ptr ;　/＊　解放すべきメモリ領域を指すポインタ　＊/

機　能　calloc, malloc, または realloc 関数によって割りつけられたメモリを解放します。
ptr は解放すべきメモリ領域を指します。

戻り値　正常終了のときは0を返します。
エラーが発生した場合は0以外の値を返します。

参照関数　calloc, malloc, realloc

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define B_SIZE          1024


main()
{
        int     i;
        char    *p;

        for (i = 1; i < 1024; i++) {
                printf("%4d block ", i);
                if ((p = malloc(i * B_SIZE)) == NULL) {
                        printf("確保できませんでした。¥n");
                        break;
                }
                printf("確保しました。");
                free(p);
                printf("; 解放しました。¥n");
        }
        exit(0);
}
```

注：無効な ptr（calloc, malloc, realloc 関数によって割りつけられたのではない領域を指すポインタ）を指定すると、その後の割つけに影響を及ぼし、エラーを起こす場合があります。

frename

レベル3

書式

```
#include <basic.h>

int frename(org, target);
char  *org;  /* 変更前ファイル名格納領域へのポインタ  */
char  *target;  /* 変更後ファイル名格納領域へのポインタ  */
```

機能

orgが指すファイルを、targetが指すファイル名に変更します。

戻り値

成功すると0を、失敗すると−1を返します。

freopen

レベル2

書式

\# include <stdio. h>

FILE *freopen(pathname, type, stream);
char *pathname; /* 新しいファイルのパス名を指すポインタ */
char *type; /* アクセスモード格納領域へのポインタ */
FILE *stream; /* FILE構造体へのポインタ */

機能

streamで指定されたストリームをクローズし、pathnameで指定したファイルを入出力ストリームに新たに割りつけます。
つまりstdin、stdout、stderr、stdaux、stdprnといったあらかじめオープンされているファイルを再割りつけするために使用します。
新たに再割りつけされたファイルはtypeで指定した型でオープンされます。
typeの値とその内容はfopen関数と同じです。

なお、stdin、stdoutに対してfreopenを行うと入出力のリダイレクトが無効になりますので注意してください。

戻り値

新しくオープンしたストリームのポインタを返します。
エラーが発生した場合は、もとのストリームがクローズされ、NULLが返されます。

参照関数

fwrite, fread, fopen

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        int     c;

        if (argc > 1) {
                if (freopen(argv[1], "r", stdin) == NULL) {
                        perror(argv[1]);
                        exit(1);
                }
        }
        while ((c = fgetchar()) != EOF)
                putchar(c);
        exit(0);
}
```

frexp

レベル２

書　　式　#include <math. h>

double frexp(x, expptr) ;
double x ;　/*　浮動小数点値　*/
int　*expptr ;　/*　指数を格納する領域へのポインタ　*/

機　　能　浮動小数点値 x を仮数 m（$x = 2^n \times m$）と指数 n に分けます。

戻 り 値　仮数 m を返します。
m の範囲は、$0.5 \leqq |m| < 1$ になります。
指数 n は expptr が示す位置に格納されます。
x が 0 の場合は、仮数、指数ともに 0 になります。

参照関数　ldexp, modf

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        double  y;
        int     exp;
        int     i;

        for (i = -10; i <= 10; i++) {
                x = (double)i / 10.0;
                y = frexp(x, &exp);
                printf("(%+4.1f) = %g * 2^%d¥n", x, y, exp);
        }
}
```

fscanf

レベル2

書　式

```
#include <stdio.h>

int fscanf (stream, format_string [, argument. . . ] ) ;
FILE  *stream ;  /*  FILE構造体へのポインタ  */
char  *format_string ;  /*  書式文字列へのポインタ  */
```

機　能

streamで指定されたストリームからデータを読み込み、format_stringの書式に従って変換し、後に続くargumentの指すフィールドに代入します。
format_stringの内容はscanf関数と同じです。
また、引数argumentはformat_stringの各変換指定に対応する型変数のポインタです。

戻り値

代入したフィールドの数を返します。
代入しなかったフィールドは含みません。
代入したフィールドがない場合は0を返します。
読み込みの前に入力が尽きてしまった場合はEOFを返します。

参照関数

fprintf, scanf, sscanf

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        void    sample(FILE *);
        FILE    *fp;

        if (argc == 1)
                sample(stdin);
        else {
                if ((fp = fopen(argv[1], "r")) == NULL) {
                        perror(argv[1]);
                        exit(1);
                }
                sample(fp);
                (void)fclose(fp);
        }
        exit(0);
}


static void     sample(FILE *fp)
{
        char    str[256];
        char    c;
        long    l;
        float   d;

        fscanf(fp, "%s¥n", str);
        fscanf(fp, "%c¥n", &c);
        fscanf(fp, "%ld¥n", &l);
        fscanf(fp, "%g", &d);
        printf("%s %c %ld %g¥n", str, c, l, d);
}
```

fseek

レベル2

書　式

```
#include <stdio.h>

int fseek(stream, offset, prev);
FILE  *stream;  /*  FILE構造体へのポインタ  */
long offset;  /*  起点からのオフセット  */
int prev;  /*  ファイルポインタの起点  */
```

機　能

streamで指定したストリームのファイルポインタをprevで指定した起点からoffsetバイト分移動します。
prevの値と内容を以下に示します。

prevの値	内　　容
SEEK_SET	ファイルの先頭
SEEK_CUR	ファイルポインタの現在位置
SEEK_END	ファイルの最後

テキストモードのストリームに対して、以下のいずれかの条件に一致しないfseekを行ってはいけません。

・offsetが0
・offsetがftell関数で返された値であり、なおかつprevがSEEK_SET

バイナリモードのストリームの場合は、ファイルの先頭より前でなければ、どこへ移動してもかまいません。
このとき、ファイルの最後より後へ移動すると、ファイルの最後の位置から移動後の位置まで、ファイルに0を書き足します。

戻り値

ファイルポインタが正しく移動したときには0を返します。
エラーのときは0以外を返します。
ターミナルやプリンタなどの、シークできないデバイスがストリームに対応している場合には戻り値は不定となります。

参照関数

ftell, lseek, rewind

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define LEN             100L


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     c;
        long    i, j;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "rb")) == NULL) {
                perror(file);
                exit(1);
        }
        (void)fseek(fp, 0L, SEEK_END);
        j = ftell(fp);
        for (i = j - LEN + 1; i <= j; i++) {
                if (fseek(fp, i, SEEK_SET) != 0)
                        break;
                else {
                        c = fgetc(fp);
                        putchar(c);
                }
        }
        (void)fclose(fp);
        exit(0);
}
```

fsetpos

レベル2

書式

```
#include <stdio.h>

int fsetpos(stream, pos) ;
FILE   *stream ;  /*  FILE構造体へのポインタ  */
fpos_t   *pos ;  /*  移動したいファイル位置を格納してあるポインタ  */
```

機能

streamが指すストリームの現在のアクセス位置を、posが指すポインタに格納されている位置に移動します。
異なる処理系の間で移植などを行う場合、処理系の違いなどにより、ファイルのアクセス位置を表す情報に違いがあるため、fseekやftell関数では十分なアクセスが行えない場合があります。
fsetposは、そのような問題を解消し、移植しやすいプログラムを記述する為に用意されています。
XCではfpos_t型は、stddef. hで以下のように型宣言されています。

```
typedef long     fpos_t ;
```

posが指すポインタに格納されているファイル位置は、同一のファイルstreamに対して行われたfgetpos関数によって格納された値ではなくてはなりません。

戻り値

成功すると0を、失敗すると0以外を返します。

参照関数

fgetpos, ftell, fseek

fstat

レベル2

書式 #include <stat. h>

```
int fstat(handle, buffer) ;
int handle ;  /*  該当ファイルのファイルハンドル  */
struct stat  *buffer ;  /*  stat 構造体を指すポインタ  */
```

機能 handleで指定されるオープン中のファイルに関する情報をbufferが指す構造体に格納します。

stat構造体はstat. hで以下のように定義されています。

```
struct stat   {
        dev_t     st_dev;    /*  ファイルならばドライブ番号、デバイスな
                                 らハンドラ番号  */
        ino_t     st_ino;    /*  常に0  */
        short     st_mode;   /*  ファイルタイプとファイル属性
                                 デバイスならばS_IFCHRのビットがセッ
                                 トされ、ファイルならばS_IFREGのビッ
                                 トがセットされる
                                 アクセスモードによってユーザー（リー
                                 ド／ライト）のビットがセットされる  */
        short     st_nlink;  /*  常に1  */
        short     st_uid;    /*  常に0  */
        short     st_gid;    /*  常に0  */
        dev_t     st_rdev;   /*  ファイルならばドライブ番号、デバイスな
                                 らハンドラ番号  */
        off_t     st_size;   /*  ファイルの大きさ（バイト数）  */
        time_t    st_atime;  /*  最終変更日時。time関数の返り値と同じ表
                                 現  */
        time_t    st_mtime;  /*  最終変更日時。time関数の返り値と同じ表
                                 現  */
        time_t    st_ctime;  /*  最終変更日時。time関数の返り値と同じ表
                                 現  */
} ;
```

戻り値 ステータス情報を正しく得た場合には0を返します。

エラーが発生した場合は－1を返し、EBADFをerrnoにセットします。

参照関数　access, chmod, filelength, stat

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <stdlib.h>
#include        <stat.h>
#include        <io.h>

main(int argc, char *argv[])
{
        struct stat     inf;
        char    *file;
        int     fn;

        file = (argc > 1) ? argv[1] : "file";
        if ((fn = open(file, O_RDONLY)) == -1) {
                perror(file);
                exit(1);
        }

        (void)fstat(fn, &inf);
        printf("name = %s, size = %d¥n", file, inf.st_size);

        close(fn);
        exit(0);
}
```

注：handle がデバイスのとき、stat 構造体の st_size と st_atime、st_mtime、st_ctime のメンバは意味をもちません。

ftell

レベル 2

書式

```
#include <stdio.h>

long ftell(stream) ;
FILE  *stream ;  /*  FILE構造体へのポインタ  */
```

機能

streamで指定したストリームの現在のファイルポインタを調べます。
ファイルポインタは、ファイルの先頭からのバイト数で示されます。

戻り値

現在のファイルポインタを返します。
オープン時のtypeが"a"、"at"以外のオープンでストリームをオープンした直後は0を返します。
エラーが発生したときは−1を返します。
ターミナルやプリンタなどのシークできないデバイスがストリームに対応している場合には戻り値は不定となります。

参照関数

fseek, lseek, tell

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     c;
        long    i, j;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "rb")) == NULL) {
                perror(file);
                exit(1);
        }
        (void)fseek(fp, 0L, SEEK_END);
        printf("file end -> %ld¥n", ftell(fp));
        (void)fclose(fp);
        exit(0);
}
```

ftime

レベル2

書　　式

#include <timeb. h>

void ftime(timeptr) ;
struct timeb　*timeptr ;　/*　timeb構造体を指すポインタ　*/

機　　能

現在の時刻をtimeptrが指す構造体timebに格納します。
構造体timebはtimeb. hで定義され、time, millitim, timezone, dstflagの4つのメンバからなります。
timeにはグリニッジ標準時1970年1月1日00:00:00から経過した秒数がセットされます。
millitimには、常に0がセットされます。
timezoneには、グリニッジ標準時と地方時との差（西回り）が分単位でセットされます。
timezoneの値は、グローバル変数timezoneの値からとります。
dstflagは、グローバル変数daylightの値により、夏時間調整が地方時に有効な場合、0以外の値となります。

戻り値

戻り値はありません。

参照関数

asctime,　ctime,　gmitime, time, tzset

プログラム例

```
#include        <stdio.h>
#include        <timeb.h>


main()
{
        struct timeb    tbuf;

        ftime(&tbuf);
        printf("time->%ld¥n", tbuf.time);
        printf("timezone->%d¥n", tbuf.timezone);
        printf("dstflag->%d¥n", tbuf.dstflag);
}
```

ftpack　　レベル 2

書式

```
#include <io.h>

int ftpack (dtlist) ;
char  *dtlist; /*  年月日時分秒の格納領域へのポインタ  */
```

機能

dtlist で指定された年月日時分秒の情報を、日時情報に変換します。
年月日時分秒の情報は、dtlist の指す領域に以下の通りに格納します。

*(dtlist + 0) : 1980 年からの相対年数(0〜99)
*(dtlist + 1) : 月(1〜12)
*(dtlist + 2) : 日(1〜31)
*(dtlist + 3) : 時(0〜23)
*(dilist + 4) : 分(0〜59)
*(dilist + 5) : 秒(0〜59)

したがって、年月日時分秒の情報を格納する領域は 6 バイト必要です。

戻り値

変換した日時情報を以下に示すフォーマットで返します。

ビット 31〜25 : 1980 年からの相対年数(0〜99)
ビット 24〜21 : 月(1〜12)
ビット 20〜16 : 日(1〜31)
ビット 15〜11 : 時(0〜23)
ビット 10〜5 : 分(0〜59)
ビット 4〜0 : 秒/2(0〜29)

ftpack によって変換した日時情報は、chgft で使用することができます。

参照関数

ftunpk, chgft, getft

プログラム例

```
#include        <stdio.h>
#include        <time.h>
#include        <io.h>
#include        <fcntl.h>
#include        <stat.h>

static char     dtlist[6] = {1990 - 1980, 3, 14, 12, 30, 0};


main()
{
        long    datetime;
        int     fn;

        if ((fn = open("¥¥CONFIG.SYS", O_RDWR)) == -1)
                puts("オープンできません。");

        datetime = ftpack(dtlist);

        chgft(fn, datetime);
}
```

ftunpk

レベル 2

書式

```
#include <io.h>

void ftuspk(datetime, dtlist) ;
int datetime ;    /* 日時情報 */
char *dtlist ;    /* 年月日時分秒の格納領域へのポインタ */
```

機能

datetime で指定された日時情報を年月日時分秒に変換し、dtlist の指す領域に格納します。

datetime のフォーマットは以下の通りです。

ビット 31～25 ： 1980 年からの相対年数(0～99)
ビット 24～21 ： 月(1～12)
ビット 20～16 ： 日(1～31)
ビット 15～11 ： 時(0～23)
ビット 10～5 ： 分(0～59)
ビット 4～0 ： 秒/2(0～29)

また、dtlist の指す領域には以下の通りに年月日時分秒の情報が格納されます。

*(dtlist + 0) ： 1980 年からの相対年数(0～99)
*(dtlist + 1) ： 月(1～12)
*(dtlist + 2) ： 日(1～31)
*(dtlist + 3) ： 時(0～23)
*(dtlist + 4) ： 分(0～59)
*(dtlist + 5) ： 秒(0～59)

したがって、年月日時分秒の情報が格納される領域は 6 バイト必要です。

getft で求めた日時情報を、ftunpk によって年月日時分秒に変換することができます。

戻り値

戻り値はありません。

参照関数

ftpack, chgft, getft

プログラム例

```
#include        <stdio.h>
#include        <time.h>
#include        <io.h>
#include        <fcntl.h>
#include        <stat.h>


main()
{
        long    datetime;
        int     fn;
        char    dtlist[6];

        if ((fn = open("¥¥CONFIG.SYS", O_RDWR)) == -1)
                puts("オープンできません。");
        datetime = getft(fn);

        ftunpk(datetime, dtlist);

        printf("%d/", dtlist[0] + 1980);
        printf("%d/", dtlist[1]);
        printf("%d ", dtlist[2]);
        printf("%d:", dtlist[3]);
        printf("%d:", dtlist[4]);
        printf("%d¥n", dtlist[5]);
}
```

fwrite

レベル2

書　　式

\# include <stdio. h>

size_t fwrite(buffer, size, count, stream) ;
void *buffer ; /* 書き込むデータ格納領域へのポインタ */
size_t size ; /* データ項目のサイズ */
size_t count ; /* 書き込みデータ項目数 */
FILE *stream ; /* FILE構造体へのポインタ */

機　　能

bufferに格納されているsizeバイトの長さのデータ項目をcountの数だけ、streamで指定されたストリームに書き込みます。
ファイルポインタは、書き込んだバイト数だけ進みます。
ファイルがテキストモードでオープンされている場合は、改行'¥n'は復帰改行'¥r¥n'に変換されます。
この変換はファイルポインタや戻り値には影響を与えません。

戻 り 値

実際に書き込んだデータ項目の数を返します。
エラーが発生した場合には、countより少ない数が返されます。

参照関数

fread, write

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define SIZE    1024


main(int argc, char *argv[])
{
        char    buf[SIZE];
        char    *file;
        FILE    *fp;
        int     n;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "rb")) == NULL) {
                perror(file);
                exit(1);
        }
        while ((n = fread(buf, sizeof(char), SIZE, fp)) > 0)
                fwrite(buf, sizeof(char), n, stdout);
        (void)fclose(fp);
        exit(0);
}
```

gcvt

レベル2

書　　式

<code># include <stdlib. h></code>

```
char  *gcvt (value, ndig, buffer) ;
double value ;  /*  変換すべき浮動小数点値  */
int ndig ;  /*  桁数  */
char  *buffer ;  /*  変換文字列の格納領域を指すポインタ  */
```

機　　能

浮動小数点値 value を ndig の桁数で printf の%f フォーマット、すなわち指数のない形式に変換し、それが不可能な場合には%e フォーマット、すなわち指数を含んだ形式の文字列に変換し、buffer の指す領域に格納します。
文字列の末尾にはヌル文字'¥0'が付加されます。

戻 り 値

変換された文字列へのポインタを返します。

参照関数

atof, atoi, ecvt, fcvt

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define PI      3.14159265358979323 85


main()
{
        char    buf[256];
        int     n;

        for (n = 1; n <= 100; n++) {
                (void)gcvt(PI / n, 10, buf);
                printf("PI / %d = %s¥n", n, buf);
        }
}
```

get

レベル 3

書式

```
#include <graph.h>

void get (x1, y1, x2, y2, buf, size) ;
int x1 ; /*  始点 X 座標  */
int y1 ; /*  始点 Y 座標  */
int x2 ; /*  終点 X 座標  */
int y2 ; /*  終点 Y 座標  */
void  *buf ; /*  画像データを読み込む領域へのポインタ  */
int size ; /*  画像のバイト数  */
```

機能

グラフィック画面の長方形の画像データを、size バイト数分、buf の指す領域に読み込みます。

x1 には始点 X 座標を指定します。
y1 には始点 Y 座標を指定します。
x2 には終点 X 座標を指定します。
y2 には終点 Y 座標を指定します。
始点と終点は、window 関数で指定したクリッピング座標の中でなければなりません。
buf には、読み込む画像データを格納する領域へのポインタを指定します。
この領域は、size バイト分なければいけません。
size には、画像データのバイト数を指定します。
画面モードにより、以下のバイト数を指定します。

16 色モード：(総ドット数+1) ÷2
256 色モード：総ドット数
65536 色モード：総ドット数×2

戻り値

戻り値はありません。

参照関数

put, window

プログラム例

```
#include        <stdio.h>
#include        <basic0.h>
#include        <graph.h>


main()
{
        static char     buf[14015 + 1];
        FILE    *fp;

        b_init();
        screen(1, 3, 1, 1);
        vpage(1);
        symbol(3, 1, "x68000", 2, 2, 2, 65535, 0);
        box(0, 0, 147, 49, 65535, 0xffff);

        get(1, 1, 146, 48, buf, sizeof(buf));

        fp = fopen("a:test.dat", "w");
        fwrite(buf, 1, 14016, fp);
        (void)fclose(fp);
        b_exit(0);
}
```

getc, getchar　レベル2

書　　式

```
#include <stdio.h>

int getc(stream) ;
FILE  *stream ;  /*  FILE構造体へのポインタ  */
int getchar( ) ;
```

機　　能

getc は stream で指定するストリームから1文字読み込みます。
getchar は getc（stdin）と同じです。

戻 り 値

読み込んだ文字を返します。
エラーが発生した場合またはエンドオブファイル（EOF）を読み込んだ場合は、EOF を返します。
どちらによる EOF かは、ferror または feof 関数を使用して調べることができます。

参照関数

fgetc, fgetchar, getch, getche, putc, putchar, ungetc

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        FILE    *fp;
        int     c;

        if (argc == 1) {
                while ((c = getchar()) != EOF)
                        putchar(c);
        }
        else {
                if ((fp = fopen(argv[1], "r")) == NULL) {
                        perror(argv[1]);
                        exit(1);
                }
                while ((c = getc(fp)) != EOF)
                        putchar(c);
                (void)fclose(fp);
        }
        exit(0);
}
```

注：getc, getchar は fgetc, fgetchar 関数と同じですが、マクロ定義と関数の両方があります。マクロ定義で使いたい場合は、stdio. h をインクルードする前に次の1行を追加してください。

```
# define MACRO
```

getch　レベル 2

書　式　#include <conio. h>

int getch( ) ;

機　能　コンソールから１文字読み込みます。
入力エコーバックはありません。

戻り値　読み込んだ文字を返します。

プログラム例

```
#include        <stdlib.h>
#include        <conio.h>


main()
{
        char    *msg = "スペースキーを押してください。";

        while (cputs(msg), getch() != ' ')
                cputs("¥r¥n");
        cputs("¥r¥nOK¥r¥n");
        exit(0);
}
```

getche

レベル２

書　　式　# include <conio. h>

int getche( ) ;

機　　能　コンソールから１文字読み込みます。
入力エコーバックします。

戻 り 値　読み込んだ文字を返します。

参照関数　cgets, getch, getchar

プログラム例

```
#include        <stdlib.h>
#include        <conio.h>
#include        <ctype.h>

main()
{
        char    *msg = "0～9の番号を選んでください。";
        int     c;

        while (cputs(msg), (c = getche()), !isdigit(c))
                cputs("¥r¥n");
        cprintf("¥r¥nselected -> [%c]¥r¥n", c);
        exit(0);
}
```

getcwd　レベル2

書　式

```
#include <direct.h>

char    *getcwd(pathbuf, n);
char    *pathbuf;  /*  パス名を格納する領域を指すポインタ  */
int n;  /*  パス名の最大長  */
```

機　能

カレントディレクトリのフルパス名をpathbufが指す領域に格納します。
パス名の最大長はnバイトで、パス名（最後のヌル文字を含めた）の長さがnの値を超えると、エラーになります。
pathbufはNULLでもかまいません。
その場合、nの大きさのバッファをmalloc関数によって自動的に割りつけ、そこにパス名を格納します。
このバッファは後に、free関数でgetcwdの戻り値（割りつけたバッファのポインタ）を指定して解放することができます。

戻り値

ディレクトリのフルパス名へのポインタを返します。
エラーのときはNULLを返し、errnoに次の値がセットされます。

errnoの値	内　　容
ENOMEN	nバイト分のメモリがない（NULLを引数として指定したとき）
ERANGE	パス名がn文字より長くなっている

参照関数　chdir, mkdir, rmdir

プログラム例

```
#include        <stdio.h>
#include        <direct.h>

#define SIZE    1024

static char     buf[SIZE];


main()
{
        if (getcwd(buf, SIZE) == NULL)
                perror("getcwd error");
        else
                printf("cwd is '%s'¥n", buf);
}
```

getenv

レベル２

書　　式　# include 〈stdlib. h〉

char　＊getenv(name)；
char　＊name；　/＊　環境変数名を指すポインタ　＊/

機　　能　環境変数リストから name で指定される環境変数名の内容を獲得します。

getenv で返された環境変数リストのエントリを直接変更してはいけません。
エントリを変更する場合には、putenv 関数を使用します。
また、環境変数リストに影響を与えずに返された文字列を加工したい場合は、strdup, strcpy などの関数を使って文字列のコピーを作成して、コピーに対して加工してください。
getenv, putenv 関数は環境変数リストをアクセスするためにグローバル変数 environ を使用します。

戻 り 値　環境変数の内容へのポインタを返します。
環境変数を発見できなかった場合は NULL を返します。

参照関数　putenv

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define NAME            "path"


main()
{
        char    *p;

        if ((p = getenv(NAME)) == NULL)
                printf("%s is not found.¥n", NAME);
        else
                printf("%s = %s¥n", NAME, p);
}
```

getfa

レベル2

書式

```
#include <io.h>

int getfa(pathname);
char  *pathname;  /*  属性を調べたいファイルのパス名へのポインタ  */
```

機能

pathnameで指定されたファイルの属性を調べます。

戻り値

正常終了ならば指定されたファイルの属性を返します。
属性のフォーマットは以下の通りです。

ビット0：読み込み専用
ビット1：隠しファイル
ビット2：システムファイル
ビット3：ボリューム名
ビット4：ディレクトリ
ビット5：通常ファイル

なお、ビット3～5までのいずれのビットも付いていない場合は通常ファイルです。

エラーならば－1を返します。

参照関数

chgfa

プログラム例

```
#include        <stdio.h>
#include        <io.h>

#define A_RDO           0x01    /* 読み込み専用 */
#define A_HDN           0x02    /* 隠しファイル */
#define A_SYS           0x04    /* システムファイル */
#define A_VOL           0x08    /* Volumeラベル */
#define A_DIR           0x10    /* ディレクトリ */
#define A_FIL           0x20    /* 通常ファイル */


main()
{
        int     r;

        if ((r = getfa("¥¥CONFIG.SYS")) == -1)
                puts("失敗しました。");
        else {
                if (r & A_RDO)
                        puts("読み込み専用");
                if (r & A_HDN)
                        puts("隠しファイル");
                if (r & A_SYS)
                        puts("システムファイル");
                if (r & A_VOL)
                        puts("Volumeラベル");
                if (r & A_DIR)
                        puts("ディレクトリ");
                if (r & A_FIL)
                        puts("通常ファイル");
        }
}
```

getft

レベル2

書　式

<io. h> を include（#include <io. h>）

```
#include <io. h>

int getft (file) ;
int file ;  /*  作成日時を調べたいファイルのファイルハンドル  */
```

機　能

fileで指定されたファイルの作成日時を調べます。

戻り値

正常終了ならば、指定されたファイルの作成日時を返します。
作成日時のフォーマットは以下の通りです。

ビット31～25　：1980年からの相対年数(0～99)
ビット24～21　：月(1～12)
ビット20～16　：日(1～31)
ビット15～11　：時(0～23)
ビット10～5　：分(0～59)
ビット4～0　：秒/2(0～29)

エラーならば−1を返します。
なお、作成日時は負の数となる場合もあるので、エラーかどうかを単に戻り値の符号で判別できないことに注意してください。
この返り値は、ftunpkによって年月日時分秒に変換することができます。

参照関数

chgft, ftunpk

プログラム例

```
#include        <stdio.h>
#include        <io.h>
#include        <fcntl.h>
#include        <stat.h>


main()
{
        int     date;
        int     fn;

        if ((fn = open("¥¥CONFIG.SYS", O_RDWR)) == -1)
                puts("オープンできません。");

        if ((date = getft(fn) == -1L)
                puts("失敗しました。");
        else
                puts(dtasc(date >> 16));
        close(fn);
}
```

getl

レベル２

書　式

```
#include <stdio.h>

int getl (fp);

FILE *fp; /* ファイル構造体へのポインタ */
```

機　能

fpで指定されたストリームからlongサイズ分読み込みます。

戻り値

読み込んだlongの値を返します。
エラーが発生した場合またはエンドオブファイル（EOF）を読み込んだ場合は、EOFを返します。
ただし、ストリームから読み込んだデータがEOFと一致している可能性がありますので、エラーとエンドオブファイルの判定はferrorとfeof関数を使用して判定してください。

参照関数

putw, putl, getw

getmem, rlsmem, getml, rlsml

レベル2

書　　式

```
#include <stdlib.h>

void  *getmem(size) ;  /*  メモリ領域の確保  */
size_t size ;  /*  確保するメモリ領域のサイズ  */

int rlsmem(p, size) ;  /*  メモリ領域の解放  */
char  *p ;  /*  解放する領域へのポインタ  */
size_t size ;  /*  解放するメモリ領域のサイズ  */

void  *getml(size) ;  /*  メモリ領域の確保  */
long size ;  /*  確保するメモリ領域のサイズ  */

int rlsml(p, size) ;  /*  メモリ領域の解放  */
char  *p ;  /*  解放する領域へのポインタ  */
long size ;  /*  解放するメモリ領域のサイズ  */
```

機　　能

メモリ領域の確保、または解放を行います。
getmem は、size バイトの領域を確保し、その領域を指すポインタを返します。
メモリが不足して領域を確保できないときには NULL を返します。
rlsmem は、getmem で確保したメモリ領域を解放します。
getml, rlsml は、引数の型が long である他は、それぞれ getmem, rlsmem と同じです。

rlsmem で指定するサイズ size は、getmem で指定したものと等しくなければなりません。getmem 以外の関数で確保した領域を解放しようとした場合、rlsmem は－1 を返します。

戻 り 値

処理が正常に終了した場合、getmem, getml は確保したメモリ領域へのポインタを、rlsmem, rlsml は 0 を返します。
エラーが発生した場合には、getmem, getml は NULL を、rlsmem, rlsml は－1 を返します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define B_SIZE  1024


main()
{
        int     n;
        char    *p;

        for (n = 1; n <= 1000; n++) {
                printf("%dKbyte: ", n);
                if ((p = getmem(n * B_SIZE)) != NULL) {
                        printf("確保できました。");
                        rlsmem(p, n * B_SIZE);
                        printf(": 全メモリを解放しました¥n");
                }
                else {
                        printf("確保できませんでした。¥n");
                        break;
                }
        }
}
```

getpid

レベル2

書式　#include <process. h>

int getpid( ) ;

機能　呼び出しプロセスを識別するプロセスIDを返します。
プロセスIDには、プログラム実行開始時にそのプロセスに固有な数値が与えられます。

戻り値　プロセスIDが返ります。

参照関数　mktemp

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <process.h>


main()
{
        char    name[256];

        sprintf(name, "¥¥tmp¥¥file%04d", getpid());
        printf("tmporary file name = %s¥n", name);
}
```

gets　　レベル2

書式　#include <stdio. h>

char　*gets(buffer) ;
char　*buffer ;　/*　読み込んだ文字を格納する領域へのポインタ　*/

機能　標準入力から改行文字'¥n'が現われるか、標準入力がエンドオブファイル（EOF）になるまで文字列を読み込みます。
読み込んだ文字列の末尾にはヌル文字'¥0'が付加され、buffer が指す領域に格納されます。

戻り値　読み込んだ文字列へのポインタを返します。
エラーが発生した場合またはエンドオブファイル（EOF）を読み込んだ場合は、NULL を返します。
どちらによる NULL かは、ferror または feof 関数を使用して調べることができます。

参照関数　fgets, fputs, puts

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define SIZE    4096

static char     buf[SIZE];


main()
{
        while (gets(buf) != NULL)
                puts(buf);
        if (ferror(stdin)) {
                perror("read error");
                abort();
        }
        exit(0);
}
```

getw

レベル2

書　式

include <stdio. h>

int getw(fp) ;
FILE　*fp; /*　FILE構造体へのポインタ　*/

機　能

fpで指定されたストリームからshortサイズ分読み込みます。

戻り値

読み込んだshortの値を返します。
エラーが発生した場合またはエンドオブファイル（EOF）を読み込んだ場合は、EOFを返します。
どちらによるEOFかはferrorまたはfeof関数を使用して調べることができます。

参照関数

putw, putl, getl

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <io.h>

main()
{
        short   s;
        FILE    *fp;

        fp = stdin;

        if ((s = getw(fp)) == EOF) {
                if (ferror(fp)) {
                        perror("fail");
                        clearerr(fp);
                }
                else if (eof(fp))
                        printf("end of file¥n");
                exit(1);
        }
        printf("number = %d¥n", s);
        exit(0);
}
```

gmtime　レベル2

書　式

```
#include <time.h>

struct tm  *gmtime(time);
time_t  *time;  /*  時刻データの格納領域を指すポインタ  */
```

機　能

時刻timeをグリニッジ標準時に変換し、tm構造体に格納します。
時刻timeはグリニッジ標準時1970年1月1日0時0分0秒から経過した秒数で、timeを呼び出すことにより得られます。
tm構造体は、time.hで以下のように宣言されています。

```
struct tm {
        int tm_sec;      /*  秒(0～59)  */
        int tm_min;      /*  分(0～59)  */
        int tm_hour;     /*  時間(0～23)  */
        int tm_mday;     /*  日付(1～31)  */
        int tm_mon;      /*  月(0～11、0が1月)  */
        int tm_year;     /*  年(現在の年から1900を引いた数)  */
        int tm_wday;     /*  曜日(0～6、0が日曜日)  */
        int tm_yday;     /*  1年を通した日付(0～365、0は1月1日)  */
        int tm_isdst;    /*  夏時間調整が無効の場合は0、
                             有効の場合は0以外  */
};
```

gmtimeは、内部の静的な領域へのポインタを返します。
したがって、gmtimeが呼び出されるたびに同じ領域に上書きされ、前の内容は失われます。

戻り値

tm構造体を指すポインタを返します。

参照関数

asctime, ctime, localtime, time

プログラム例

```
#include        <stdio.h>
#include        <time.h>


main()
{
        struct tm       *l_tm;
        struct tm       *u_tm;
        time_t  ltime;

        time(&ltime);

        u_tm = gmtime(&ltime);
        printf("ただ今の UCT 時刻は %s¥n", asctime(u_tm));

        l_tm = localtime(&ltime);
        printf("ただ今の 現地時刻は %s¥n", asctime(l_tm));
}
```

hantozen

レベル2

書式

```
#include <jfctype.h>

int hantozen(c);
int c;  /* 変換する文字（半角文字） */
```

機能

半角文字cを全角文字に変換します。
cは0x20≦c≦0×7E、または0xA1≦c≦0xDFの範囲内になければなりません。
カナはカタカナに変換されます。
cのビット16が1のときには、ひらがなに変換されます。

戻り値

シフトJIS漢字コードでの全角文字を返します。
変換できなかった場合にはcを返します。

参照関数

zentohan

home

レベル3

書　式

#include <graph. h>

```
void home(pa, x, y) ;
int pa ;  /*  グラフィックページ  */
int x ;  /*  左端X座標  */
int y ;  /*  上端Y座標  */
```

機　能

グラフィック画面の表示位置を設定します。
pa にはグラフィック画面のページを指定します（0～3)。
ただし、画面のモードにより、最大ページ数が異なります。
x には左端X座標、y には上端Y座標を指定します。
x、yもまた、画面のモードにより最大値が異なります(0～511、または0～1023)。

戻り値

戻り値はありません。

プログラム例

```
#include        <basic0.h>
#include        <graph.h>


main()
{
        b_init();

        screen(2, 0, 1, 1);
        apage(0);
        vpage(1);
        window(0, 0, 1023, 1023);
        circle(511, 511, 100, 15, 0, 360, 256);
        paint(511, 511, 9);

        home(0, 255, 255);

        b_exit(0);
}
```

hsv

レベル 3

書　式

```
#include <graph.h>

int hsv(hu, sa, va);
int hu;  /*  色相  */
int sa;  /*  濃さ  */
int va;  /*  明るさ  */
```

機　能

色相 hu, 濃さ sa, 明るさ va に応じた色コードを得ます。
なお、色コードは 0～65534 の偶数となります。
それぞれの指定可能範囲は以下の通りです。

hu : 0～192
sa : 0～31
va : 0～31

戻り値

色コードを返します。

プログラム例

```
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i, j, k;
        int     kyu(int, int, int);

        b_init();
        screen(1, 3, 1, 1);
        vpage(1);
        for (i = 0; i < 16; i++) {
                for (j = 0; j < 4; j++) {
                        for (k = 0; k < 3; k++) {
                                kyu(j * 128, k * 170,
                                 i * 12 + j * 3 + k);
                        }
                }
        }
        b_exit(0);
}


static int      kyu(int x, int y, int h)
{
        int     i, j, k;
        int     c;

        fill(x, y, x + 127, y + 169, 0);
        for (i = 0; i < 32; i++) {
                j = 31 - i;
                for (k = 0; k < 2; k++) {
                        c = hsv(h, 31, i) + k;
                        circle(x + 64, y + 64,
                         j * 2 + k, c, 0, 360, 256);
                        paint(x + 64, y + 64, c);
                }
        }
}
```

hypot

レベル 2

書　　式

include <math. h>

double hypot(x, y) ;
double x, y ;　/*　直角三角形の 2 辺　*/

機　　能

2 辺 x, y の長さから直角三角形の斜辺の長さを計算します。
計算式は sqrt (x×x+y×y) と同じです。

戻 り 値

直角三角形の斜辺の長さを返します。
オーバーフローが発生した場合は、errno に ERANGE がセットされ返り値は HUGE_VAL になります。
またエラー処理は、matherr 関数を使って行うことができます。

参照関数

cabs

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        double  y;
        int     i;
        int     j;

        for (i = 1; i <= 32; i++) {
                for (j = 1; j <= 32; j++) {
                        x = (double)i;
                        y = (double)j;
                        printf("hypot(%+4.1f, %+4.1f) = %g¥n",
                         x, y, hypot(x, y));
                }
        }
}
```

img_color

レベル3

書　式

```
#include <image.h>

void img_color(hu_cal);
char hu_cal;  /*  色調モード  */
```

機　能

画像とり込みを行う際のとり込み画像の色調補正（グラフィックパレットの補正）を行います。

hu_cal には次の色調モードを指定します。

0 ： 16 色モード
1 ： 256 色モード
2 ： 65536 色モード

戻り値

戻り値はありません。

img_home　レベル3

書式

```
#include <image.h>

void img_home(x, y, dir, cnt, wt) ;
int   x, y ;
char  dir ;
int   cnt, wt ;
```

機能

グラフィック画面の表示位置を移動します。
x には始点の X 座標（0～511）、y には始点の Y 座標（0～511）を指定します。
dir, cnt, wt はそれぞれ、移動方向（1～9）、移動回数（0～65535）、待ち時間（0～65535）です。

戻り値

戻り値はありません。

img_ht

レベル 3

書　式

```
#include <image.h>

void img_ht(x1, y1, x2, y2, p, a) ;
int x1;  /*  始点X座標  */
int y1;  /*  始点Y座標  */
int x2;  /*  終点X座標  */
int y2;  /*  終点Y座標  */
char p;  /*  グラフィック画面の機能設定  */
char a;  /*  半透明表示指定  */
```

機　能

グラフィック画面の指定範囲とテレビ画面との半透明表示の制御を行います。
x1 には始点 X 座標を指定します。
y1 には始点 Y 座標を指定します。
x2 には終点 X 座標を指定します。
y2 には終点 Y 座標を指定します。
p にはグラフィック画面の指定領域への半透明表示の設定有無を指定します。

0 ：半透明表示を設定しない
1 ：半透明表示を設定する

a には半透明表示の実行の有無を指定します。

0 ：半透明表示を実行しない
1 ：半透明表示を実行する

本関数実行前に、screen 関数または crt 関数により、スーパーインポーズの低解像度モード（512×512、または 256×256）にしておく必要があります。
また、x1、y1、x2、y2 は実画面サイズにより、512×512 のときは、0～511 まで、256×256 のときは、0～255 までを指定することができます。

戻り値

戻り値はありません。

img_load

レベル 3

書　式　#include <image.h>

```
int img_load(fname, x, y, b) ;
char  *fname ; /*  入力ファイル名へのポインタ  */
int x ; /*  ロードするグラフィック画面の X 座標  */
int y ; /*  ロードするグラフィック画面の Y 座標  */
int b ; /*  ロード終了位置の設定  */
```

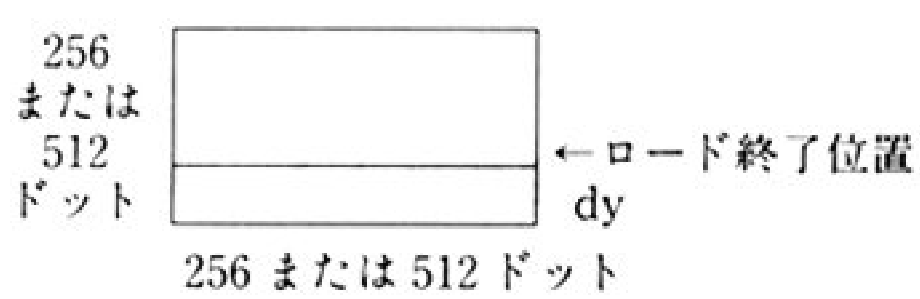
機　能　fname が指すファイルのデータを、グラフィック画面へ読み込みます。
ファイルの拡張子は、解像度により次のように決まっています。

拡張子	画面のモード	色
GS0	256×256	16 色
GM0	256×256	256 色
GL0	256×256	65536 色
GS3	512×512	16 色
GM3	512×512	256 色
GL3	512×512	65536 色

b : ロード終了位置

$$b=\frac{dy}{n}$$

dy…下図の通り Y 座標の下からのドット数



n…拡張子

GS0 ……32 ──→ $0 \leqq b < 7$
GS3 ……16
GM0 ……16
GM3 ……8
GL0 ……8
(GS3〜GL0) $0 \leqq b \leqq 7$
GL3 ……使用不可

戻り値　正常終了のときは 0 を、異常終了のときは 0 以外を返します。

参照関数　img_save

img_pos

レベル3

書　　式

```
#include <image.h>

void img_pos(h_cal) ;
char h_cal;  /*  画面水平位置補正データ  */
```

機　　能

画像とり込み時の画面水平位置の補正を行います。
h_cal は、0～255 までの範囲で指定します。

戻 り 値

戻り値はありません。

img_put

レベル 3

書　式

```
#include <image.h>

void img_put(x1, y1, x2, y2, na, len) ;
int  x1, y1, x2, y2, len ;
void   *na ;
```

機　能

グラフィックデータを表示します。
(x1, y1) で表される座標から (x2, y2) で表される座標までの間、8ドットごとに移動しながら、na が指すパターンデータを表示します。
表示するパターンデータの大きさは 256×256 ドットです。
表示データは、131,072 バイト必要です。

戻り値

戻り値はありません。

img_save

レベル3

書　式

```
#include <image.h>

int img_save(fname, x, y) ;
char  *fname ;  /*  出力ファイル名へのポインタ  */
int x ;  /*  セーブするグラフィック画面のX座標  */
iny y ;  /*  セーブするグラフィック画面のY座標  */
```

機　能

グラフィック画面のデータを、fnameが指すファイルへ書き出します。
ファイルの拡張子は、解像度により次のように決まっています。

拡張子	画面のモード	色
GS0	256×256	16色
GM0	256×256	256色
GL0	256×256	65536色
GS3	512×512	16色
GM3	512×512	256色
GL3	512×512	65536色

戻り値

正常終了のときは0を、異常終了のときは0以外を返します。

参照関数

img_load

img_scrn　レベル 3

書　式

```
#include <image.h>

void img_scrn(size, mode, disp);
char size, mode, disp;
```

機　能

グラフィック画面のモードを設定します。
引数はそれぞれ次のように指定します。

size（表示画面サイズ）
0：256×256ドット
1：512×512ドット
2：768×512ドット

mode（実画面および色モード）

0：	1024×1024ドット	16色
1：	512×512ドット	16色
2：	512×512ドット	256色
3：	512×512ドット	65536色

disp（ディスプレイ解像度）
0：標準解像度
1：高解像度

戻り値

戻り値はありません。

img_set

レベル3

書　　式

```
#include <image.h>

void img_set(img_ctl0[, img_ctl1, img_ctl2, img_ctl3, img_ctl4]) ;
char img_ctl0;  /*  制御符号  */
int img_ctl1, img_ctl2, img_ctl3, img_ctl4;  /*  制御符号  */
```

機　　能

カラーイメージユニット(CZ-6VT1)の制御を行います。
img_ctlには、カラーイメージユニットのコントロールコードを指定します(0x00～0xFF)。

戻 り 値

戻り値はありません。

img_still　レベル3

書式

```
#include <image.h>

void img_still(ch) ;
char ch; /*　モード　*/
```

機能

画像とり込みの設定を行います。
ch には画像とり込みのモードを指定します。

0 ：実行（静止画）
1 ：待機（動画）

本関数の実行前に、screen 関数により低解像モード（512×512、または 256×256）にしておく必要があります。
また、画像とり込みを行うプログラムの終了時には、必ず img_still(0) を実行してください。

戻り値

戻り値はありません。

instr

レベル３

書　　式

```
#include <basic.h>

int instr(i, str1, str2) ;
int i;  /*  比較を開始する位置  */
char  *str1;  /*  文字列  */
char  *str2;  /*  文字列  */
```

機　　能

str1のi番目の文字から始めて、str2と等しい部分を探します。
iに指定する値は、先頭を1として何番目かを数値で指定します。

戻 り 値

str1にstr2と等しい部分のあった位置を返します。
返される値は、先頭を1とした数値を返します。
なければ0を返します。

isalkana, ispnkana, isalnmkana, isprkana, isgrkana

レベル 2

書式

```
#include <jfctype.h>

int isalkana(c) ;  /*  英文字またはカナ文字の判定  */
int ispnkana(c) ;  /*  英句読点またはカナ句読点の判定  */
int isalnmkana(c) ;  /*  英数字またはカナ文字の判定  */
int isprkana(c) ;  /*  表示可能文字（空白を含む）の判定  */
int isgrkana(c) ;  /*  表示可能文字（空白を除く）の判定  */
int c;  /*  判定する文字  */
```

機能

これらの関数はcで与えられた文字の判定を行います。
isalkanaはcが英文字またはカナ文字であるか否かを判定します。
ispnkanaはcが英句読点またはカナ句読点であるか否かを判定します。
isalnmkanaはcが英数字またはカナ文字であるか否かを判定します。
isprkanaはcが表示可能文字（空白を含む）であるか否かを判定します。
isgrkanaはcが表示可能文字（空白を除く）であるか否かを判定します。

戻り値

条件を満たせば0以外の値を、満たさなければ0を返します。

参照関数

isalnum, isprint, iskana, iskpun, iskmoji

isupper, islower, isalpha, isdigit, isalnum, isgraph, isprint, isascii, isspace, ispunct, iscntrl, isxdigit, isodigit, iscsym, iscsymf

レベル2

書式

```
#include <ctype.h>

int isupper(c) ;  /*  大文字の判定  */
int islower(c) ;  /*  小文字の判定  */
int isalpha(c) ;  /*  英文字の判定  */
int isdigit(c) ;  /*  10進数の判定  */
int isalnum(c) ;  /*  英数字の判定  */
int isgraph(c) ;  /*  空白以外の表示可能文字の判定  */
int isprint(c) ;  /*  空白を含む表示可能文字の判定  */
int isascii(c) ;  /*  ASCII文字の判定  */
int isspace(c) ;  /*  空白文字の判定  */
int ispunct(c) ;  /*  句読点の判定  */
int iscntrl(c) ;  /*  制御文字の判定  */
int isxdigit(c) ;  /*  16進数の判定  */
int isodigit(c) ;  /*  8進数の判定  */
int iscsym(c) ;  /*  C言語の識別子として使用できる文字の判定  */
int iscsymf(c) ;  /*  C言語の識別子の先頭文字として使用できる文字の判定  */
int c;
```

機能

これらの関数はcで与えられた文字の判定を行います。
なお、これらにはマクロ定義と関数版があります。
関数版はfctype.hを、マクロ版はctype.hをインクルードして使用します。
それぞれ、cが以下の範囲内であるか否かを判定します。

isupper	英大文字	'A'～'Z' (0x41～0x5A)
islower	英小文字	'a'～'z' (0x61～0x7A)
isalpha	英文字	'A'～'Z' (0x41～0x5A)
		'a'～'z' (0x61～0x7A)
isdigit	数字	'0'～'9' (0x30～0x39)
isalnum	英文字、数字	'0'～'9' (0x30～0x39)
		'A'～'Z' (0x41～0x5A)
		'a'～'z' (0x61～0x7A)
isgraph	英文字、記号、数字（空白を除く）	(0x21～0x7E)
isprint	英文字、記号、数字（空白を含む）	(0x20～0x7E)
isascii	ASCII文字	(0x00～0x7F)

isspace	空白文字	(0x09～0x0D、0x20)
ispunct	句読点	(0x20～0x2F)
		(0x3A～0x40)
		(0x5B～0x60)
		(0x7B～0x7E)
iscntrl	制御文字	(0x00～0x1F、0x7F)
isxdigit	16 進数表記用文字	'0'～'9' (0x30～0x39)
		'A'～'F' (0x41～0x46)
		'a'～'f' (0x61～0x66)
isodigit	8 進数表記用文字	'0'～'7' (0x30～0x37)
iscsym	英文字	'A'～'Z' (0x41～0x5A)
		'a'～'z' (0x61～0x7A)
	数字	'0'～'9' (0x30～0x39)
	アンダーライン	'_' (0x5F)
iscsymf	英文字	'A'～'Z' (0x41～0x5A)
		'a'～'z' (0x61～0x7A)
	アンダーライン	'_' (0x5F)

戻り値 条件を満たせば0以外の値を、満たさなければ0を返します。

参照関数 toascii, tolower, toupper

プログラム例

```
#include        <stdio.h>
#include        <ctype.h>


main()
{
        int     c;

        for (c = 0x00; c < 0x80; c++) {
                printf("0x%04x", c);
                printf("%3s", isalnum(c) ? "AN" : "");
                printf("%3s", isalpha(c) ? "A" : "");
                printf("%3s", isascii(c) ? "AS" : "");
                printf("¥n");
        }
}
```

isatty

レベル2

書　式

#include <io. h>

int isatty(device) ;
int device; /*　ファイルハンドル　*/

機　能

deviceで指定したファイルハンドルが文字デバイス（ターミナル、コンソール、プリンタ、シリアルポートなど）であるかどうかを調べます。

戻り値

ファイルハンドルが文字デバイスの場合には0以外の値を返し、そうでなければ0を返します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <io.h>
#include        <stat.h>


main(int argc, char *argv[])
{
        char    *file;
        int     fn;

        file = (argc > 1) ? argv[1] : "con";
        if ((fn = open(file, O_RDONLY)) == -1) {
                perror(file);
                exit(1);
        }

        printf("'%s' is ", file);
        if (isatty(fn) == 0)
                printf("not character device¥n");
        else
                printf("character device¥n");

        close(fn);
        exit(0);
}
```

iskana, iskpun, iskmoji

レベル 2

書　　式　# include <jfctype. h>

```
int iskana(c) ; /*  カナ文字またはカナ句読点の判定  */
int iskpun(c) ; /*  カナ句読点の判定  */
int iskmoji(c) ; /*  カナ文字の判定  */
int c; /*  判定する文字  */
```

機　　能　これらの関数はcで与えられた文字の判定を行います。
iskana はcがカナ文字またはカナ句読点であるか否かを判定します。
iskpun はcがカナ句読点であるか否かを判定します。
iskmoji はcがカナ文字であるか否かを判定します。

戻 り 値　条件を満たせば0以外の値を、満たさなければ0を返します。

参照関数　isalpha, islower, isupper, isxdigit, isdigit, ispunct, isspace, isalnum, isalkana, ispnkana, isalnmkana, isprkana, zentohan

プログラム例

```
#include        <stdio.h>
#include        <jfctype.h>


main()
{
        int     c;

        /* 1byteカナ文字だけ出力するフィルタ */
        while ((c = getchar()) != EOF) {
                if (iskanji(c))
                        (void)getchar();
                else if (iskana(c))
                        putchar(c);
        }
}
```

iskanji, iskanji2　レベル2

書式

```
#include <jfctype.h>

int iskanji(c) ;  /*  漢字の1バイト目の判定  */
int iskanji2(c) ;  /*  漢字の2バイト目の判定  */
int c;  /*  判定する文字  */
```

機能

これらの関数はcで与えられた文字の判定を行います。
iskanjiはcがシフトJISコードの1バイト目であるか否かを判定します。
iskanji2はcがシフトJISコードの2バイト目であるか否かを判定します。

シフトJISコードの1バイト目……0x81≦c≦0x9F または 0xE0≦c≦0xEF
シフトJISコードの2バイト目……0x40≦c≦0x7F または 0x80≦c≦0xFC

戻り値

条件を満たせば0以外の値を、満たさなければ0を返します。

参照関数

jiszen, jisl0, jisl1, jisl2

itoa

レベル 2

書　式

include <stdlib. h>

char　*itoa(value, string, radix) ;
int value ;　/*　変換すべき整数値　*/
char　*string ;　/*　変換した文字列を格納する領域を指すポインタ　*/
int radix ;　/*　基数　*/

機　能

valueで指定した整数値を文字列（最大33バイト）に変換し、stringの指す領域に格納します。

文字列の末尾にはヌル文字'¥0'が付加されます。

radixはvalueの基数で、2～36の範囲で進数を指定します。

もしradixが10でvalueの値が負の場合は、文字列の先頭にマイナス記号（－）が付加されます。

戻り値

変換された文字列を指すポインタを返します。

参照関数

ltoa, ultoa, wtoa

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        char    buf[34];
        int     i;

        for (i = 0; i < 100; i++) {
                printf("%2d : ", i);
                printf("%7s ", itoa(i, buf, 2));
                printf("%3s ", itoa(i, buf, 8));
                printf("%2s ", itoa(i, buf, 10));
                printf("%2s ", itoa(i, buf, 16));
                printf("¥n");
        }
}
```

jisalpha, jisupper, jislower, jisdigit, jiskata, jishira, jiskigou, jisspace

レベル2

書　式

```
#include <jfctype.h>

int jisalpha(c) ;  /*  全角英文字の判定  */
int jisupper(c) ;  /*  全角英大文字の判定  */
int jislower(c) ;  /*  全角英小文字の判定  */
int jisdigit(c) ;  /*  全角数字の判定  */
int jiskata(c) ;  /*  全角カタカナ文字の判定  */
int jishira(c) ;  /*  全角ひらがな文字の判定  */
int jiskigou(c) ;  /*  全角句読点文字の判定  */
int jisspace(c) ;  /*  全角空白文字の判定  */
int c;  /*  判定する文字  */
```

機　能

これらの関数はcで与えられたシフトJISコードでの文字の判定を行います。
jisalphaはcが全角英文字であるか否かを判定します。
jisupperはcが全角英大文字であるか否かを判定します。
jislowerはcが全角英小文字であるか否かを判定します。
jisdigitはcが全角数字であるか否かを判定します。
jiskataはcが全角カタカナ文字であるか否かを判定します。
jishiraはcが全角ひらがな文字であるか否かを判定します。
jiskigouはcが全角句読点文字であるか否かを判定します。
jisspaceはcが全角空白文字であるか否かを判定します。

戻り値

条件を満たせば0以外の値を、満たさなければ0を返します。

参照関数

jiszen, jisl0, jisl1, jisl2, iskanji, iskanji2

jiszen, jisl0, jisl1, jisl2 レベル2

書　式　#include <jfctype. h>

int jiszen(c) ; /*　全角文字の判定　*/
int jisl0(c) ; /*　全角空白文字、JIS 第一水準記号の判定　*/
int jisl1(c) ; /*　JIS 第一水準漢字の判定　*/
int jisl2(c) ; /*　JIS 第二水準漢字の判定　*/
int c; /*　判定する文字　*/

機　能　これらの関数はcで与えられたシフトJISコードでの文字の判定を行います。
jiszen はcが全角文字であるか否かを判定します。
jisl0 は全角空白文字または JIS 第一水準の記号（非漢字）であるか否かを判定します。
jisl1 はcが JIS 第一水準漢字であるか否かを判定します。
jisl2 はcが JIS 第二水準漢字であるか否かを判定します。

戻り値　条件を満たせば0以外の値を、満たさなければ0を返します。

参照関数　iskanji, iskanji2

プログラム例

```
#include        <stdio.h>
#include        <jfctype.h>


main()
{
        unsigned short  ch;
        int     c;
        int     cc;

        while ((c = getchar()) != EOF) {
                if (iskanji(c)) {
                        cc = c;
                        ch = (c << 8);
                        ch += (c = getchar());
                        printf("%c%c は、", cc, c);
                        if (jisl0(ch))
                                printf("非漢字¥n");
                        else if (jisl1(ch))
                                printf("第1水準¥n");
                        else if (jisl2(ch))
                                printf("第2水準¥n");
                        else
                                printf("1byte文字¥n");
                }
        }
}
```

jstrchr

レベル2

書　式

```
#include <jstring.h>

unsigned char  *jstrchr(string, c);
unsigned char  *string; /*  サーチが行われる文字列へのポインタ  */
int c; /*  サーチする文字  */
```

機　能

stringが指す文字列の中で、文字cが最初に現れる位置を調べます。
本関数は、strchr関数の漢字対応版であり、文字列にシフトJISコードの漢字を含むことができるという点を除けば、基本的な機能は同じです。
漢字は1文字として扱われ、シフトJISコードの2バイト目がヌル文字'¥0'であれば1バイト目もヌル文字であると解釈されます。
サーチする文字が漢字の場合は、シフトJISコードを使用し、このコードの1バイト目を上位バイトに、2バイト目を下位バイトに設定します。
サーチする文字が半角の場合には、下位8ビットを使用し、上位バイトはゼロにしておきます。

戻り値

文字cが現れた位置を指すポインタを返します。
文字cが文字列にないときはNULLを返します。

参照関数

strchr, jstrrchr

jstrcmp　レベル2

書　　式　#include <jstring.h>

```
int jstrcmp(string1, string2) ;
unsigned char  * string1 ;  /*  文字列1へのポインタ  */
unsigned char  * string2 ;  /*  文字列2へのポインタ  */
```

機　　能　本関数はstrcmp関数の漢字対応版であり、文字列にシフトJISコードの漢字を含むことができるという点を除けば、基本的な機能は同じです。
漢字は1文字として扱われ、シフトJISコードの2バイト目がヌル文字'¥0'であれば1バイト目もヌル文字であると解釈されます。
なお、文字列の比較は次の大小関係によって行われます。

英数文字＜カナ文字＜漢字

戻 り 値　文字列1と文字列2の大小関係を返します。
戻り値とその内容は次の通りです。

戻り値	内　　容
負	string1はstring2より小さい
0	string1とstring2は等しい
正	string1はstring2より大きい

参照関数　strcmp

jstrncmp

レベル２

書式

```
#include <jstring.h>

int jstrncmp(string1, string2, n) ;
unsigned char  * string1 ;  /*  文字列１へのポインタ  */
unsigned char  * string2 ;  /*  文字列２へのポインタ  */
int n ;  /*  比較する文字数  */
```

機能

文字列の比較を行います。
本関数はstrncmp関数の漢字対応版であり、文字列にシフトJISコードの漢字を含むことができるという点を除けば、基本的な機能は同じです。
nは比較する文字数を指定するのでありバイト数を指定するのではありません。
したがって、漢字は１文字として数えます。
また、シフトJISコードの２バイト目がヌル文字'¥0'ならば、１バイト目もヌル文字であると解釈します。
なお、jstrncmp関数は、次の大小関数により比較を行います。

英数文字＜カナ文字＜漢字

戻り値

文字列１と文字列２の大小関係を返します。
戻り値とその内容は次の通りです。

戻り値	内　容
負	string1はstring2より小さい
0	string1とstring2は等しい
正	string1はstring2より大きい

参照関数　strncmp

jstrrchr

レベル 2

書　式

```
#include <jstring.h>

unsigned char  *jstrrchr(string, c) ;
unsigned char  *string ;  /*  サーチが行われる文字列へのポインタ  */
int c ;  /*  サーチする文字  */
```

機　能

string1 が指す文字列の中で、文字 c が最後に現れた位置を調べます。
本関数は、strrchr 関数の漢字対応版であり、文字列にシフト JIS コードの漢字を含むことができるという点を除けば、基本的な機能は同じです。
漢字は 1 文字として扱われ、シフト JIS コードの 2 バイト目がヌル文字'¥0'であれば 1 バイト目もヌル文字であると解釈されます。
サーチする文字が漢字の場合は、シフト JIS コードを使用し、このコードの 1 バイト目を上位バイトに、2 バイト目を下位バイトに設定します。
サーチする文字が半角の場合には、下位 8 ビットを使用し、上位バイトはゼロにします。

戻り値

文字 c が最後に現れた位置を指すポインタを返します。
文字 c が文字列にないときは NULL を返します。

参照関数

strrchr, strpbrk, jstrchr

kbhit

レベル２

書　式

#include <conio. h>

int kbhit( ) ;

機　能

コンソールからの入力があるかどうかを調べます。
入力を調べるのみで、入力は行いません。

戻り値

入力があるときは0以外を返し、ないときは0を返します。

プログラム例

```
#include        <stdio.h>
#include        <conio.h>

main()
{
        puts("何かキーを押してください。");

        while (kbhit() == 0)
                ;
}
```

key　レベル3

書式

```
#include <basic0.h>

void key(n, s) ;
int n; /*　ファンクションキー番号　*/
char *s; /*　文字列　*/
```

機能

再定義可能なファンクションキーに文字列を定義します。
nにはファンクションキーの番号を指定します。(1～20)。
sには文字列を指定します。
ファンクションキーに定義できる文字列の長さは31バイトまで有効です。
定義内容に32バイト以上を指定した場合はエラーになり、何もしません。
BASICからCに変換したときにKEY文がkey関数に展開されますが、BASICのKEY文とは動作が異なりますので注意してください。
制御コードは、@を前につけたA～Z, [,¥,],],^,_ (¥は¥¥にすること) で表します。(例. "@A")。
"@@"は引用符 (") を表します。

戻り値

戻り値はありません。

keysns

レベル3

書　式　#include <image.h>

```
int keysns( );
```

機　能　キーボードの入力状態を調べます。

戻り値　入力状態によって、以下の値を返します。

入力あり：−1
入力なし：　0

ldexp

レベル 2

書　　式　#include <math. h>

double ldexp(x, exp) ;
double x ;
int exp ;

機　　能　x × 2^{exp}を求めます。

戻 り 値　x × 2^{exp}を計算した結果を返します。
オーバーフローが発生した場合は、errno に ERANGE がセットされ、返り値は、x と同じ符号がついた HUGE_VAL になります。
アンダーフローが発生した場合は、errno に ERANGE がセットされ、返り値は、x と同じ符号がついた 0 になります。

参照関数　frexp, modf

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        int     i;
        int     j;

        for (i = -32; i <= 32; i++) {
                for (j = -32; j <= 32; j++) {
                        x = (double)i / 10.0;
                        printf("ldexp(%+4.1f, %d) = %g¥n",
                         x, j, ldexp(x, j));
                }
        }
}
```

ldiv

レベル2

書　式

include <stdlib. h>

ldiv_t ldiv(number, denom) ;
long number ; /* 割られる値 */
long denom ; /* 割る値 */

機　能

number を denom で割った商と余りを ldiv_t 型の構造体として返します。
ldiv は符号付き整数の除算を行う場合に、処理系の違いによる結果の相違を解消するために用意されています。
ldiv_t 型構造体は、符号付き整数2つをメンバに持つ構造体で、stddef.h で以下のように型宣言されています。

```
typedef struct {
        long    quot ;
        long    rem ;
} ldiv_t ;
```

戻り値

number を denom で割った商と余りを ldiv_t 型の構造体として返します。
戻り値は、構造体の実体であることに注意してください。
構造体メンバ quot には商を、メンバ rem には余りを返します。

参照関数

div

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        ldiv_t  ans;
        long    num;
        long    denom;

        for (num = -10L; num <= 10L; num++) {
                for (denom = -10L; denom <= 10L; denom++), {
                        if (denom == 0L)
                                continue;
                        ans = ldiv(num, denom);
                        printf("%ld / %ld = %ld 余り %ld¥n",
                         num, denom, ans.quot, ans.rem);
                }
        }
}
```

line

レベル3

書式

```
#include <graph.h>

void line (x1, y1, x2, y2, p, ls) ;
int x1 ; /*  始点X座標  */
int y1 ; /*  始点Y座標  */
int x2 ; /*  終点X座標  */
int y2 ; /*  終点Y座標  */
int p ; /*  パレットコード  */
int ls ; /*  ラインスタイル  */
```

機能

グラフィック画面に直線を描画します
x1には始点X座標を指定します（−32768〜32767)。
y1には始点Y座標を指定します（−32768〜32767)。
x2には終点X座標を指定します（−32768〜32767)。
y2には終点Y座標を指定します（−32768〜32767)。
pにはパレットコードを指定します（0〜65535)。
画面モードにより最大値が異なります。
lsにはラインスタイルを指定します（0〜65535)。

戻り値

戻り値はありません。

参照関数

screen

プログラム例

```
#include        <stdlib.h>
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i;
        int     x1, x2, y1, y2;
        int     pc;
        int     ls;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        for (i = 0; i < 500; i++) {
                x1 = rand() % 768;
                y1 = rand() % 512;
                x2 = rand() % 768;
                y2 = rand() % 512;
                pc = rand() % 16;
                ls = rand() * 2 + 1;
                line(x1, y1, x2, y2, pc, ls);
        }
        b_exit(0);
}
```

localtime

レベル 2

書　　式

```
#include <time.h>

struct tm   *localtime(time) ;
time_t   *time ;  /*  時刻データの格納領域を指すポインタ   */
```

機　　能

時刻timeを地方時に変換し、tm構造体に格納します。
時刻timeはグリニッジ標準時1970年1月1日0時0分0秒から経過した秒数で、time関数を呼び出すことにより得られます。
tm構造体はtime.hで以下のように宣言されています。

```
struct tm {
        int   tm_sec ;      /*  秒(0～59)   */
        int   tm_min ;      /*  分(0～59)   */
        int   tm_hour ;     /*  時間(0～23)   */
        int   tm_mday ;     /*  日付(1～31)   */
        int   tm_mon ;      /*  月(0～11、0が1月)   */
        int   tm_year ;     /*  年(現在の年から1900を引いた数)   */
        int   tm_wday ;     /*  曜日(0～6、0が日曜日)   */
        int   tm_yday ;     /*  1年を通した日付(0～365、0は1月1日)   */
        int   tm_isdst ;    /*  夏時間調整が無効の場合は0、
                                有効の場合は0以外   */
} ;
```

localtimeは、内部の静的な領域へのポインタを返します。
したがって、localtimeが呼び出されるたびに同じ領域に上書きされ、前の内容は失われます。
それ以外にも、ctime関数を呼び出すと、ctime関数の内部でlocaltimeを呼び出すので、それ以前に呼び出された、localtimeで返された内容は失われます。

戻 り 値

tm構造体を指すポインタを返します。

参照関数

asctime, ctime, gmtime, time

locate

レベル3

書　式

```
#include <basic0.h>

void locate(x, y) ;
int x; /* X座標 */
int y; /* Y座標 */
```

機　能

テキスト座標のx、yで指定した位置にカーソルを移動します。

戻り値

戻り値はありません。

log, log10

レベル 2

書　　式　# include <math. h>

double log(x) ; /*　自然対数　*/
double log10(x) ; /*　底が 10 の対数　*/
double x ;

機　　能　log は x の e を底とする自然対数を、log10 は x の 10 を底とする常用対数を求めます。

戻 り 値　log は $\log_e X$ を、log10 は $\log_{10} X$ を返します。
引数が負の数などの不正な値の場合は errno に EDOM がセットされ、返り値は不定になります。
またエラー処理は、matherr 関数を使って行うことができます。

参照関数　exp, matherr, pow

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        int     i;

        for (i = 1; i <= 30; i++) {
                x = (double)i / 10.0;
                printf("log(%+4.1f) = %g¥n", x, log(x));
        }
        for (i = 1; i <= 30; i++) {
                x = (double)i / 10.0;
                printf("log10(%+4.1f) = %g¥n", x, log10(x));
        }
}
```

longjmp

レベル2

書　式

```
#include <setjmp.h>

void longjmp(env, value);
jmp_buf  env;  /*  setjmpによりセーブされたスタック環境  */
int value;  /*  setjmpが返す（かのように見える）戻り値  */
```

機　能

setjmp関数でセーブしたスタック環境を復元して、対応するsetjmp関数の呼び出し元へ制御を移します。
envはsetjmp.hで定義されたjmp_buf型の変数であり、あらかじめsetjmp関数によってスタック環境がセーブされていなければなりません。
valueは、setjmp関数の呼び出し元へ、setjmp関数の戻り値として渡される値です。
必ず0以外の値を指定してください。
もし0を指定すると、強制的に1を指定したことになります。

戻り値

longjmpは、setjmp関数を呼び出す前に呼び出してはいけません。
また、対応するsetjmp関数の呼び出し元の関数がreturn文を実行した後や、関数から抜け出したりした後でlongjmpをしてはいけません。
ほかに、setjmp関数呼び出し時とlongjmp呼び出し時でスーパーバイザ/ユーザーモードが異なっていてはいけません。
もし、上記の禁止事項が守られていない状態でlongjmpを行うと、その後のプログラムの動作は保障されません。
setjmp関数の呼び出し元へ制御を移すので、longjmpの呼び出し元へはリターンしません。

setjmp関数を呼び出すルーチンにおけるレジスタ変数は、longjmpを実行した後では、正しい値として使用することはできません。

参照関数

setjmp

lseek　レベル2

書　式

```
#include <io.h>

long lseek(handle, offset, origin) ;
int handle ; /*  ファイルのファイルハンドル  */
long offset ; /*  起点からのオフセット（移動バイト数）  */
int origin ; /*  ファイルポインタの起点  */
```

機　能　handleで指定されたファイルのファイルポインタをoriginで指定した起点からoffsetバイト分移動します。
originの値と内容を以下に示します。

originの値	内　容
SEEK_SET	ファイルの先頭
SEEK_CUR	ファイルポインタの現在の位置
SEEK_END	ファイルの最後

ファイルの先頭よりも前へ移動することはできません。
ファイルの最後より後へ移動すると、ファイルの最後の位置から移動後の位置まで、ファイルに0を書き足します。

戻り値　新しいファイルポインタの位置を先頭からのバイト数で返します。
エラーが発生した場合は−1を返し、errnoに次の値がセットされます。

errnoの値	内　容
EBADF	ファイルが無効
EINVAL	originの値が無効、またはoffsetで指定された位置がファイルの先頭位置よりも前を示している

シークが不可能なデバイス（ターミナルやプリンタなど）では、戻り値は不定となります。

参照関数　fseek, tell

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <io.h>
#include        <stat.h>

#define LEN             100L


main(int argc, char *argv[])
{
        char    buf[LEN];
        char    *file;
        int     fn;

        file = (argc > 1) ? argv[1] : "file";
        if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
                perror(file);
                exit(1);
        }
        if (lseek(fn, -LEN, SEEK_END) == -1L) {
                perror("seek error");
                exit(1);
        }
        else {
                read(fn, buf, LEN);
                write(1, buf, LEN);
        }
        close(fn);
        exit(0);
}
```

ltoa

レベル 2

書　式

```
#include <stdlib.h>

char  *ltoa(value, string, radix) ;
long value ;  /*  変換すべき long 型整数値  */
char  *string ;  /*  変換した文字列を格納する領域を指すポインタ  */
int radix ;  /*  基数  */
```

機　能

value で指定された long 型整数値を文字列（最大 33 バイト）に変換し、string が指す領域に格納します。

文字列の末尾にはヌル文字'¥0'が付加されます。

radix は value の基数で、2 から 36 の範囲で進数を指定します。

もし radix が 10 で value が負の場合は、文字列の先頭にマイナス記号（—）が付加されます。

戻り値

変換された文字列を指すポインタを返します。

参照関数

itoa, ultoa, wtoa

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        char    buf[34];
        long    l;
        int     i;

        for (i = -10; i < 10; i++) {
                l = (long)i * 10000L;
                printf("%7ld: ", l);
                printf("%032s ", ltoa(l, buf, 2));
                printf("%11s ", ltoa(l, buf, 8));
                printf("%7s ", ltoa(l, buf, 10));
                printf("%8s ", ltoa(l, buf, 16));
                printf("¥n");
        }
}
```

malloc　レベル2

書式　#include <stdlib. h>

void　*malloc(size)；
size_t size；　/*　割りつけるメモリ領域のサイズ　*/

機能　sizeバイトのメモリ領域を割りつけます。

戻り値　割りつけた領域を指すポインタを返します。
割りつけられた領域は、どんな型のデータにも利用できるようになっています。
メモリに空きがないときはNULLを返します。

参照関数　calloc, free, realloc

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define B_SIZE          1024


main()
{
        int     i;
        char    *p;

        for (i = 1; i < 1024; i++) {
                printf("%4d block ", i);
                if ((p = malloc(i * B_SIZE)) == NULL) {
                        printf("確保できませんでした。¥n");
                        break;
                }
                printf("確保しました。");
                free(p);
                printf("; 解放しました。¥n");
        }
        exit(0);
}
```

m_alloc

レベル3

書式

```
#include <music2.h>

int m_alloc(t, size);
char t;  /*  トラック番号  */
int size;  /*  トラックサイズ  */
```

機能

FM 音源および MIDI のためのトラックバッファを確保します。
全トラックバッファ内のデータはすべてクリアされます。
t にはトラック番号を指定します (1～223)。
size にはトラックサイズをバイト数で指定します (1～65536)。

戻り値

正常終了の場合は 0 を返します。
引数のエラーなどの場合エラー番号として、0 以外の正の数を返します。
FM 音源が使用不可の場合、−1 を返します。
なお、エラー番号とその意味は以下の通りです。

番号	エラーの意味
−1	音楽用関数は無効です
1	テンポの指定が無効です
2	トラック番号が無効です
3	music cmd error
4	サイズの指定が無効です
5	メモリー確保できません
6	チャンネル番号が無効です
7	配列の指定に誤りがあります
8	音色番号が無効です
9	音楽用関数は無効です
19	MML の文法に誤りがあります
20	[ ] の中の指定に誤りがあります
21	] がありません
22	繰り返しの値が無効です
23	繰り返し番号の指定がありません
24	繰り返し番号の指定が無効です
25	オクターブの番号がありません
26	オクターブの指定が無効です

番号	エラーの意味
27	長さの値が無効です
28	トラック容量が足りません
29	音色番号の指定がありません
30	@W の値がありません
31	@W の値が無効です
32	テンポの値がありません
33	テンポの値が無効です
34	長さの値がありません
35	音量の値がありません
36	音量の値が無効です
37	キーコードの値がありません
38	キーコードの値が無効です
39	音色番号が無効です
40	} がありません
41	{ } の中に音符が多すぎます
42	{ } の中に音符がありません
43	Q の指定に誤りがあります
44	{ } の中の文法に誤りがあります
45	Y の指定に誤りがあります
46	@L の指定に誤りがあります
47	P の指定に誤りがあります
48	タイの指定に誤りがあります
49	MIDI ボードがありません
50	MIDI モードになっています
51	MIDI データに誤りがあります
52	MIDI 数値データが長過ぎます
53	MIDIMML に誤りがあります
54	MIDI チャンネルが無効です
55	MIDI 音色の番号が無効です
56	レジスタの指定が無効です
57	9_ch 目以降は使えません

以上のエラー番号の説明は、以下の関数においても同様です。

m_assign, m_cont, m_play, m_stop, m_tempo, m_trk, m_vget, m_vset
md_cont, md_off, md_on, md_play, md_stop, md_wrt

m_assign

レベル3

書式

#include <music2. h>

```
int m_assign(channel, track) ;
char channel ;  /*  FM 音源のチャンネル  */
char track ;  /*  トラック番号  */
```

機能

FM 音源または MIDI のチャンネルにトラックを割りあてます。
デフォルトでは、1～8 チャネルがそれぞれ 1～8 トラックに対応しています。
また、MIDI ボードが本体に実装されている場合、9～24 チャンネルはそれぞれデフォルトで 81～96 トラックに対応しています。
9～24 のチャンネルは MIDI ボードのためのチャンネルで、MIDI ボードがなければ使用できません。

戻り値

正常終了の場合は 0 を返します。
引数のエラーなどの場合エラー番号として 0 以外の正の数を返します。
FM 音源が使用不可の場合、－1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

matherr

レベル2

書　式

```
#include <math.h>

int matherr(x);
struct exception  *x;  /*  exception構造体を指すポインタ  */
```

機　能

数値演算ライブラリの実行時に発生したエラーを処理します。
数値演算関数でエラーが発生すると、必ずmatherrが呼び出されるので、ユーザーは独自のエラー処理を行うことができます。
matherrは、math.hで以下のように宣言されている構造体exceptionへのポインタを引数に持って呼び出されます。

```
struct exception  {
        int type;                     /*  エラーの種類  */
        char   *name;                 /*  エラーの発生した関数名を指すポイン
                                          タ  */
        double arg1, arg2, retval;    /*  argnはエラーを起した引数
                                          retvalは戻り値  */
};
```

typeは数値演算時に発生したエラーの種類を示します。

typeの値	内　　容
DOMAIN	引数の領域が正しくない
SING	引数が正しくない
OVERFLOW	オーバーフローした
UNDERFLOW	アンダーフローした
TLOSS	全体的に桁落ちした
PLOSS	部分的に桁落ちした

nameはエラーをおこした関数名を指すポインタです。
関数名はヌル文字で終わっています。
arg1とarg2はエラーを起した引数の値を示しています（引数が1つの場合はarg1に格納されます）。
retvalはエラーが発生した数値演算関数のデフォルトの戻り値で、ユーザーがここに数値を代入して、それを変更することもできます。

matherr 自身の戻り値は、エラーが実際に起ったかどうかを指示しなければなりません。

matherr が 0 を返した場合は、errno に対応するエラー値をセットします。

matherr が 0 以外の値を返した場合には、errno は変化しません。

このとき、retval に数値を代入して、数値演算関数の戻り値を変更できます。

戻り値　エラー回復できなかった場合には 0 を返し、エラー回復を行うことができた場合には 0 以外の値を返さなければなりません。

参照関数　acos, asin, atan, atan2, cabs, cos, cosh, exp, hypot, log, pow, sin, sinh, sqrt, tan

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        int     matherr(struct exception *);
        double  x, y;

        x = -10.0;
        y = log(x);
}


static int      matherr(struct exception *ep)
{
        char    *s;
        switch (ep->type) {
        case DOMAIN:
                s = "引数の領域が正しくない。";
                break;
        case SING:
                s = "引数が正しくない。";
                break;
        case OVERFLOW:
                s = "オーバーフローした。";
                break;
        case UNDERFLOW:
                s = "アンダーフローした。";
                break;
        case TLOSS:
                s = "全体的に桁落ちした。";
                break;
        case PLOSS:
                s = "部分的に桁落ちした。";
                break;
        default:
                return (0);
        }
        fprintf(stderr, "%s: %s¥n", ep->name, s);
        return (0);
}
```

max

レベル２

書式

#include <stdlib. h>

(type)max(a, b);
(type)a;
(type)b;

機能

maxは、aとbを比較し、大きい方を返すマクロです。
2つの引数と戻り値の型は同じ型で、比較演算が可能な型でなければなりません。
引数には、副作用を引き起こすものや、関数などを指定すると、期待した結果と異なる場合があるので注意して下さい。

戻り値

2つの引数のうち、大きい方の値を返します。

参照関数

min

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define SIZE    10

static int      a[SIZE] = {3, 1, 4, 1, 5, 9, 2, 5, 7, 1};


main()
{
        int     m;
        int     i;

        m = a[0];
        for (i = 0; i < SIZE; i++)
                m = max(m, a[i]);
        printf("最大の値は%d¥n", m);
}
```

m_cont

レベル3

書式

```
#include <music2.h>

int m_cont(c1, c2, c3, c4, c5, c6, c7, c8);
int c1, c2, c3, c4, c5, c6, c7, c8;  /*  チャンネル  */
```

機能

FM音源のチャンネルを指定して、一時中断していた演奏を再開します。
指定するチャンネルの個数が7個以下の場合は、余りの引数に 'NASI' を指定しなければなりません。
全チャンネルが 'NASI' ならば、1～8チャンネル中現在中断していたチャンネルすべてを再開します。
チャンネルは1～8が指定できます。

戻り値

正常終了の場合は0を返します。
引数のエラーなどの場合、エラー番号として0以外の正の数を返します。
FM音源が使用不可の場合、－1を返します。
なお、エラー番号とその意味は、m_alloc関数を参照してください。

md_cont

レベル3

書式

```
#include <music2.h>

int md_cont(channel) ;
int channel ;     /* ビット指定のチャンネル */
```

機能

FM音源およびMIDIのチャンネルをビットで指定して、一時中断していた演奏を再開します。
channelに設定する値は、以下の通りです。

ビット 0～7 : 1～8チャンネル (FM音源)
ビット 8～23 : 9～24チャンネル (MIDI)
ビット 24～31 : 常に0

ビットを1にしたチャンネルの演奏を再開します。
0または'NASI'を指定すると、現在中断していたチャンネルすべてを再開します。

戻り値

正常終了の場合は0を返します。
引数のエラーなどの場合、エラー番号として0以外の正の数を返します。
FM音源を使用不可の場合、−1を返します。
なお、エラー番号とその意味は、m_alloc関数を参照してください。

md_init

レベル3

書　式

```
#include <music2.h>

int md_init();
```

機　能

MIDI ボートの初期化のみを行います。
m_assign で割りあてたトラックは、m_init で初期化してください。
MIDI ボードを使用する場合には、必ず m_init の実行直後に実行してください。
MIDI ボードを使用しない場合には、実行する必要はありません。
MIDI ボードがない場合には、エラーになります。

戻り値

正常終了の場合は 0 を返します。
MIDI ボードがない場合、エラー番号 49 を返します。
FM 音源が使用不可の場合、−1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

参照関数

m_alloc, m_init

md_off

レベル 3

書　式

```
#include <music2.h>

int md_off(channel) ;
int channel ;   /* ビット指定のチャンネル  */
```

機　能

MIDI のチャンネルをビットで指定して、MIDI ボードへの出力を停止します。
channel に設定する値は、以下の通りです。

ビット0〜7　：常に 0
ビット8〜23　：9〜24 チャンネル (MIDI)
ビット24〜31：常に 0

ビットを 1 にしたチャンネルの出力を停止します。
0 または 'NASI' を指定すると、9〜24 すべてのチャンネルの出力を停止します。
MIDI ボードがない場合には、エラーになります。

戻り値

正常終了の場合は 0 を返します。
引数のエラーなどの場合、エラー番号として 0 以外の正の数を返します。
FM 音源が使用不可の場合、−1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_on

レベル3

書　　式

```
#include <music2.h>

int md_on(channel);
int channel;   /* ビット指定のチャンネル */
```

機　　能

MIDIのチャンネルをビットで指定して、MIDIボードへの出力を開始します。
channelに設定する値は、以下の通りです。

ビット0〜7　：　常に0
ビット8〜23　：　9〜24チャンネル（MIDI）
ビット24〜31　：　常に0

ビットを1にしたチャンネルの出力を開始します。
0または'NASI'を指定すると、9〜24すべてのチャンネルの出力を開始します。
MIDIボードがない場合には、エラーになります。

戻り値

正常終了の場合は0を返します。
引数のエラーなどの場合、エラー番号として0以外の正の数を返します。
FM音源が使用不可の場合、−1を返します。
なお、エラー番号とその意味は、m_alloc関数を参照してください。

md_play

レベル３

書　　式

```
#include <music2.h>

int md_play(channel) ;
int channel ;   /* ビット指定のチャンネル */
```

機　　能

FM 音源および MIDI のチャンネルをビットで指定して、演奏を開始します。
channel に設定する値は、以下の通りです。

ビット0〜7　：1〜8 チャンネル（FM 音源）
ビット8〜23　：9〜24 チャンネル（MIDI）
ビット24〜31：常に 0

ビットを 1 にしたチャンネルの演奏を開始します。
0 または 'NASI' を指定すると、すべてのチャンネルの演奏を開始します。

戻 り 値

正常終了の場合は 0 を返します。
引数のエラーなどの場合、エラー番号として 0 以外の正の数を返します。
FM 音源が使用不可か MIDI ボードがない場合、−1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_regr

レベル3

書　式

```
#include <music2.h>

int md_regr(grpno, regno) ;
char grpno;  /*  グループ番号  */
char regno;  /*  レジスタ番号  */
```

機　能

MIDIコントローラのレジスタを読み込みます。
MIDIコントローラのgrpnoで指定したグループのregnoで指定したレジスタを読み込みます。
グループ番号grpnoは0～15を指定します。
レジスタ番号regnoは0～15を指定します。
MIDIボードがない場合や、存在しないレジスタを指定した場合にはエラーになります。

戻り値

正常終了の場合は読み込んだレジスタの内容を返します。
FM音源が使用不可かMIDIボードがない場合、−1を返します。
存在しないレジスタを指定した場合、−2を返します。

md_regw

レベル3

書式

```
#include <music2. h>

int md_regw(grpno, regno, data) ;
char grpno;  /*  グループ番号/*
char regno;  /*  レジスタ番号  */
char data;  /*  書き込むデータ  */
```

機能

MIDIコントローラのレジスタにデータを書き込みます。
MIDIコントローラのgrpnoで指定したグループのregnoで指定したレジスタにdataで指定したデータを書き込みます。
グループ番号grpnoは0〜15を指定します。
レジスタ番号regnoは0〜15を指定します。
書き込むデータdataは0〜255を指定します。
MIDIボードがない場合や、存在しないレジスタを指定した場合には、エラーになります。

戻り値

正常終了の場合は、書き込んだデータを返します。
FM音源が使用不可かMIDIボードがない場合、−1を返します。
存在しないレジスタを指定した場合、−2を返します。

md_stat

レベル3

書式

```
#include <music2.h>

int md_stat(ch) ;
int ch;  /*  チャンネル  */
```

機能

chで指定したチャンネルのMIDIボードへの出力の状態を調べます。
chには9～24を指定します。
0または'NASI'を指定すると、9～24すべてのチャンネルの状態を調べます。
MIDIボードがない場合には、エラーになります。

戻り値

正常終了の場合はチャンネルの状態を返します。
このとき、chに指定した値により以下のようになります。

・chが9～24のとき

0：停止中
1：出力中

・chが0または'NASI'のとき

ビット0～7　：　常に0
ビット8～23　：　9～24チャンネルの状態
（ビットが0で停止中、1で出力中）
ビット24～31：　常に0

FM音源が使用不可かMIDIボードがない場合、−1を返します。

md_stop　　レベル3

書　式

```
#include <music2.h>

int md_stop(channel);
int channel;  /* ビット指定のチャンネル */
```

機　能

FM 音源および MIDI のチャンネルをビットで指定して、演奏を一時中断します。

channel に設定する値は、以下の通りです。

ビット0〜7　：1〜8 チャンネル（FM 音源）
ビット8〜23　：9〜24 チャンネル（MIDI）
ビット24〜31：常に0

ビットを1にしたチャンネルの演奏を一時中断します。
0または'NASI'を指定すると、現在演奏しているチャンネルすべてを一時中断します。

戻り値

正常終了の場合は0を返します。
引数のエラーなどの場合、エラー番号として0以外の正の数を返します。
FM 音源が使用不可の場合、−1を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

md_wrt

レベル3

書　式

```
#include <music2.h>

int md_wrt(data);
char data;  /*  出力データ  */
```

機　能

MIDIボードにデータを出力します。
MIDIボードにdataで指定したデータを出力します。
出力データdataは0～255を指定します。
MIDIボードがない場合には、エラーになります。

戻り値

正常終了の場合は0を返します。
引数のエラーなどの場合、エラー番号として0以外の正の数を返します。
FM音源が使用不可の場合、−1を返します。
なお、エラー番号とその意味は、m_alloc関数を参照してください。

memccpy レベル2

書式 #include <string. h>

void ＊memccpy(dest, orig, c, size) ;
void ＊dest ; /＊ 転送先の領域を指すポインタ ＊/
void ＊orig ; /＊ 転送元の領域を指すポインタ ＊/
char c ; /＊ 転送を終了させる文字 ＊/
size_t size ; /＊ 転送バイト数 ＊/

機能 origが指す領域からdestが指す領域へ文字cが見つかるか、または転送したバイト数がsizeバイトに達するまで転送します。

戻り値 文字cまでを転送したときには、dest内のcの次の位置を指すポインタを返します。
orig内に文字cと同じ文字がなく、sizeバイト転送したときはNULLを返します。

参照関数 memchr, memcmp, memcpy, memset

プログラム例

```
#include        <stdio.h>
#include        <string.h>

#define SIZE    256

static char     sbuf[SIZE] = "hello world.good bye";
static char     dbuf[SIZE] = "";


main()
{
        char    *p;
        int     c;

        c = '.';

        if ((p = memccpy(dbuf, sbuf, c, SIZE)) != NULL)
                printf("dbuf = '%s'¥n", dbuf);
        else {
                printf("dbuf = '%s'¥n", dbuf);
                printf("return value is NULL");
        }

}
```

memchr

レベル2

書　式

include <string. h>

void　*memchr(buffer, c, size) ;
void　*buffer ; /*　検索を行う領域を指すポインタ　*/
int c ; /*　検索する文字　*/
size_t size ; /*　検索を行うバイト数　*/

機　能

bufferが指す領域からsizeバイトの範囲で文字cを検索します。

戻り値

buffer内のcの位置を指すポインタを返します。
文字cがなかった場合は、NULLを返します。

参照関数

memccpy, memcmp, memcpy, memset

プログラム例

```
#include        <stdio.h>
#include        <string.h>

#define SIZE    16

static char     buf[SIZE + 1] = "0123456789abcdef";


main()
{
        char    *p;
        int     c;

        for (c = ' '; c < 0x7f; c++) {
                printf("char = %c, ", c);
                if ((p = memchr(buf, c, SIZE)) == NULL)
                        printf("not found¥n");
                else
                        printf("offset = %d¥n", p - buf);
        }
}
```

memcmp　レベル2

書　式　# include <string. h>

int memcmp(mem1, mem2, size) ;
void　*mem1 ;　/*　比較領域1を指すポインタ　*/
void　*mem2 ;　/*　比較領域2を指すポインタ　*/
size_t size ;　/*　比較バイト数　*/

機　能　mem1が指す領域とmem2が指す領域をsizeバイトの範囲だけ比較します。

戻り値　2つの領域の内容の大小関係を表す下記の値を返します。

戻り値	内　容
負	mem1はmem2より小さい
0	mem1とmem2は等しい
正	mem1はmem2より大きい

参照関数　memccpy, memchr, memcpy, memcmpi, memicmp, memset, strcmp

プログラム例

```
#include        <stdio.h>
#include        <string.h>

#define SIZE    16

static char     mem1[SIZE + 1] = "0123456789abcdef";
static char     mem2[SIZE + 1] = "0123456789ABCDEF";


main()
{
        int     r;

        if ((r = memcmp(mem1, mem2, SIZE)) == 0)
                printf("mem1 == mem2¥n");
        else if (r < 0)
                printf("mem1 < mem2¥n");
        else if (r > 0)
                printf("mem1 > mem2¥n");
}
```

memcmpi, memicmp

レベル2

書　式

```
#include <string.h>

int memicmp(mem1, mem2, size);
int memcmpi(mem1, mem2, size);

void   *mem1;  /*  比較領域1を指すポインタ  */
void   *mem2;  /*  比較領域2を指すポインタ  */
size_t size;  /*  比較バイト数  */
```

機　能

mem1が指す領域とmem2が指す領域をsizeバイトの範囲だけ、大文字/小文字を区別しないで比較します。
memcmpi, memicmpは，大文字/小文字を区別しない以外はmemcmp関数と同じです。
また、memcmpi, memicmpは、関数名が異なるだけで、機能は同じです。

戻り値

2つの関係を表す下記の値を返します。

戻り値	内　　容
負	mem1はmem2より小さい
0	mem1とmem2は等しい
正	mem1はmem2より大きい

参照関数

memcmp, strcmpi

プログラム例

```
#include        <stdio.h>
#include        <string.h>

#define SIZE    16

static char     mem1[SIZE + 1] = "0123456789abcdef";
static char     mem2[SIZE + 1] = "0123456789ABCDEF";


main()
{
        int     r;

        if ((r = memcmpi(mem1, mem2, SIZE)) == 0)
                printf("mem1 == mem2¥n");
        else if (r < 0)
                printf("mem1 < mem2¥n");
        else if (r > 0)
                printf("mem1 > mem2¥n");
}
```

memcpy, memmove, movedata, movmem　レベル2

書　式

```
#include <string.h>

void  *memcpy(dest, source, size);
void  *memmove(dest, source, size);
void movedata(source, dest, size);
void movmem(source, dest, size);

void  *source;  /*  転送元領域のポインタ  */
void  *dest;  /*  転送先領域のポインタ  */
size_t size;  /*  転送バイト数  */
```

機　能

sourceが指す領域の内容を、destが指す領域にsizeバイト転送します。
これらの領域は、重なり合っていても、正しく転送が行われます。
ただし、XC以外の処理系では、memcpyは領域を重ねると正しく転送されない処理になっている可能性があるので、移植を行う場合には注意してください。
また、movedataは、他の処理系では引数の数が異なる場合がありますので、これも移植時には注意してください。

sourceとdestの指定順が異なるので、これらの関数を使用するときには注意してください。
またこれらの関数は、strcpy関数などとは異なり、ヌル文字'¥0'には関係なくsizeバイトを転送します。

戻り値

memcpy, memmoveは転送先領域へのポインタを返します。
movedata, movmemには戻り値はありません。

参照関数

memccpy, memchr, memcmp, memset

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     des[] = "SHARP X68000";
static char     src1[] = "Sharp";
static char     src2[] = "sharp";


main()
{
        printf("des = %s¥n", des);
        (void)memcpy(des, src1, strlen(src1));
        printf("des = %s¥n", des);
        movmem(src2, des, strlen(src2));
        printf("des = %s¥n", des);
}
```

memset, setmem

レベル2

書式

#include <string. h>

void ＊memset(mem, c, size) ;
void setmem(mem, size, c) ;
void ＊mem ; /＊ 文字を埋める領域を指すポインタ ＊/
int c ; /＊ 設定する文字 ＊/
size_t size ; /＊ 設定バイト数 ＊/

機能

mem が指す領域の size バイトを文字 c で埋めます。

戻り値

memset は文字を埋めた領域を指すポインタを返します。
setmem には、返り値はありません。

参照関数

memccpy, memchr, memcmp, memcpy

プログラム例

```
#include        <stdio.h>
#include        <string.h>

#define SIZE    256

static char     buf[SIZE];


main()
{
        void    test(char *);

        (void)memset(buf, 'x', SIZE);
        test(buf);
        /* setmem */
        setmem(buf, SIZE / 2, 'o');
        test(buf);
}


static void     test(char *p)
{
        int     i, j;

        for (i = 0; i < 16; i++) {
                for (j = 0; j < 16; j++) {
                        putchar(*p);
                        putchar(' ');
                        ++p;
                }
                putchar('¥n');
        }
        putchar('¥n');
}
```

m_free

レベル3

書式

```
#include <music.h>

int m_free(track) ;
char track ; /* トラック番号 */
```

機能

FM音源およびMIDIで使用する、trackで指定されたトラックの残り容量を調べます。

戻り値

正常終了の場合は、残り容量をバイト数で返します。
FM音源が使用不可の場合、−1を返します。
また、楽譜データの最小単位は3バイトですので、残り容量が2バイト以下の場合は、それ以上トラックに書き込むことはできません。

min

レベル2

書　　式　#include <stdlib. h>

(type)min(a, b);
(type)a;
(type)b;

機　　能　minは、aとbを比較し、値の小さい方を返すマクロです。
2つの引数と、戻り値の型は同じ型で、比較演算が可能な型でなければなりません。
引数には、副作用を引き起こすものや、関数などを指定すると、期待した結果と異なる場合があるので注意してください。

戻り値　2つの引数のうち、小さい方の値を返します。

参照関数　max

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define SIZE    10

static int      a[SIZE] = {3, 1, 4, 1, 5, 9, 2, 5, 7, 1};


main()
{
        int     m;
        int     i;

        m = a[0];
        for (i = 0; i < SIZE; i++)
                m = min(m, a[i]);
        printf("最小の値は%d¥n", m);
}
```

m_init

レベル 3

書　　式　# include 〈music2. h〉

int m_init( ) ;

機　　能　FM 音源（トラックの楽譜データ、音色）を初期化します。
m_assign で割りあてたトラックをデフォルトに戻します。

戻 り 値　正常終了の場合は 0 を返します。
FM 音源が使用不可の場合、−1 を返します。

mkdir

レベル2

書　　式　#include <direct. h>

int mkdir(pathname) ;
char　*pathname;　/*　新ディレクトリ名へのポインタ　*/

機　　能　pathname で指定した、新しいディクレトリを作成します。

戻 り 値　正常終了ならば、正の数（0も）を返します。
エラーならば−1を返します。
また、エラーが発生した場合、errno と _doserrno にそれぞれ c のエラー番号と OS のエラー番号をセットします。

参照関数　chdir, rmdir

プログラム例

```
#include        <stdio.h>
#include        <direct.h>

#define DIR1    "¥¥tmp2"
#define DIR2    "¥¥tmp2¥¥sub"


main()
{
        if (mkdir(DIR1) >= 0) {
                printf("%s is created.¥n", DIR1);
                if (mkdir(DIR2) == 0) {
                        printf("%s is created.¥n", DIR2);
                        (void)rmdir(DIR2);
                        printf("%s is removed.¥n", DIR2);
                }
                (void)rmdir(DIR1);
                printf("%s is removed.¥n", DIR1);
        }
}
```

mktemp

レベル 2

書式

```
#include <io.h>

char  *mktemp(temp) ;
char  *temp;
```

機能

引数で指定された文字列をもとに、カレントディレクトリに存在しないファイル名を作成します。
引数の temp が指す文字列は、次のように設定します。

```
char temp[ ] = "strXXXXXX";
```

"str"は新たに作成するファイル名の先頭部分となり、"XXXXXX"の部分を別の文字列に置き換えます。
なお、"XXXXXX"の部分は、'X'という文字（アスキーコード 0x58）で 6 文字以上なければならず、別の文字が混ざっていたり 6 文字に足りないときはエラーとなります。
"XXXXXX"の部分は、後ろから 5 文字分を、その時の時刻の分と秒をもとに算出した値を文字列に変換して置き換えます。
後ろから 6 文字目は、通常'0'ですが、このようにして新たに作成したファイル名がすでにカレントディレクトリに存在する場合は、次に示す規則に従って置き換えていきます。

'0'→'a'→'b'→'c'→……→'z'

したがって、例えば"tmp000044"という文字列が生成され、同じ名前のファイルがすでにカレントディレクトリに存在している場合には、mktemp は次に"tmpa00044"というファイルの存在の有無を調べ、もし無ければ、その文字例を返します。
以下に、temp に設定した文字列の変換の例を示します。

- "sampleXXXXX"→"sample000300"
- "tmpXXXXXXX"→"tmpX010100"
- "TPXXXXXX" →"TPa01234"

  （"TP001234"が存在していた場合）

なお、mktemp は引数の temp が指す文字列を直接変更するので、mktemp の呼び出しの前に temp に設定されていた文字列は失われます。
したがって、mktemp の引数には " で囲まれた文字列を指定してはいけません。

戻り値 新たにファイル名が作成できた場合には、そのファイル名の文字列を指し示すポインタを返します。
引数が不正か、新たなファイル名を作成できない場合は、エラーとなり、NULLを返します。

参照関数 fopen, getpid, open

プログラム例

```
#include        <stdio.h>
#include        <string.h>
#include        <io.h>

#define MAX             6
#define NAMLEN          16
#define TEMPLATE        "pcsXXXXXX"

static char     tnames[MAX][NAMLEN];


main()
{
        char    *make_uniq(char *);
        int     touch( char *);
        int     i, j;

        for (i = 0; i < MAX; i++) {
                if (make_uniq(tnames[i]) == NULL) {
                        puts("create error");
                        break;
                }
                else if (touch(tnames[i]) != 0) {
                        puts("touch error");
                        break;
                }
                printf("%s is created.¥n", tnames[i]);
        }
        for (j = 0; j < i; j++) {
                unlink(tnames[j]);
                printf("%s is removed.¥n", tnames[j]);
        }
}


static char     *make_uniq(char *name)
{
        char    temp[NAMLEN];
        char    *r;

        strcpy(temp, TEMPLATE);
        if ((r = mktemp(temp)) == NULL)
                return (NULL);
        strcpy(name, r);
        return (name);
}


static int      touch(char *name)
{
        FILE    *fp;

        if ((fp = fopen(name, "w")) == NULL)
                return (-1);
        (void)fclose(fp);
        return (0);
}
```

mktime

レベル2

書　式

#include <time. h>

time_t　mktime(time_ptr) ;
struct tm　＊time_ptr ;　/＊　tm 構造体に分解された時間情報　＊/

機　能

time_ptr が指す構造体に格納されているローカル時間を、time 関数が返す値と同じ形式のカレンダ時間に変換します。
また、tm 構造体のメンバ tm_wday, tm_yday は、変換する時には参照されませんが、変換後には正しい値が設定されます。
他のメンバも、正しい範囲に調整されます。

戻り値

変換に成功するとカレンダ時間を返します。
失敗すると、(−1)を time_t 型に変換した、すなわち(time_t)(−1)とキャストされた値を返します。

参照関数

time, gmtime, localtime

プログラム例

```
#include        <stdio.h>
#include        <time.h>


main()
{
        struct tm       dt;
        time_t  time1;
        time_t  time2;
        double  diff;

        dt.tm_year = 2001 - 1900;
        dt.tm_mon = 1;
        dt.tm_mday = 1;
        dt.tm_hour = 0;
        dt.tm_min = 0;
        dt.tm_sec = 0;
        dt.tm_isdst = 0;
        time1 = mktime(&dt);

        time(&time2);

        diff = difftime(time1, time2);
        printf("21世紀まであと%g秒¥n", diff);
}
```

modf

レベル 2

書　式　#include <math. h>

double modf(x, intptr) ;
double x ;　/*　浮動小数点値　*/
double *intptr ;　/*　整数部を格納する領域へのポインタ　*/

機　能　浮動小数点値 x を小数部と整数部に分けます。

戻り値　x と同じ符号の小数部を返し、x と同じ符号が付いた整数部は intptr に格納されます。

参照関数　frexp, ldexp

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        double  ix, fx;
        int     i;

        for (i = -15; i <= 15; i++) {
                x = (double)i / 10.0;
                fx = modf(x, &ix);
                printf("%+4.1f ->modf()-> %g, %g¥n", x, ix, fx);
        }
}
```

mouse

レベル3

書　　式　#include <mouse.h>

int mouse(i);
int i;　/*　機能コード　*/

機　　能　マウスの制御を行います。
iに指定できる機能コードは次の通りです。

0：マウスの初期化、ソフトキーボードの表示を許す
1：マウスカーソルの表示
2：マウスカーソルを消す
3：マウスカーソルの表示状態の通知
4：ソフトキーボードの表示をさせない

戻 り 値　iに3を指定した場合は、以下に示すマウスカーソルの表示状態を返します。

−1：表示している
0：消えている

iに0、1、2、4を指定した場合は、0を返します。
iに0〜4以外を指定した場合は、エラーとして−1を返します。

プログラム例

```
#include        <stdio.h>
#include        <string.h>
#include        <mouse.h>

#define M_INIT          0
#define M_ON            1
#define M_OFF           2
#define M_STAT          3


main(int argc, char *argv[])
{
        if (argc == 1)
                puts(mouse(M_STAT) ? "on" : "off");
        else if (argc == 2) {
                if (strcmp(argv[1], "on") == 0)
                        mouse(M_ON);
                else if (strcmp(argv[1], "off") == 0)
                        mouse(M_OFF);
                else
                        puts("Usage: mouse [on|off]");
        }
        else
                puts("Usage: mouse [on|off]");
}
```

m_play

レベル 3

書　式

```
#include <music.h>

int m_play(c1, c2, c3, c4, c5, c6, c7, c8) ;
int c1, c2, c3, c4, c5, c6, c7, c8 ;  /*  チャンネル  */
```

機　能

FM 音源の指定したチャンネルに割りあてたトラックデータを演奏開始します。
指定するチャンネルの個数が 7 個以下の場合は、余りの引数に 'NASI' を指定しなければなりません。
全チャンネルが 'NASI' ならば、1～8 チャンネルすべてを演奏開始します。
チャンネルは、1～8 が指定できます。

戻り値

正常終了の場合は 0 を返します。
引数のエラーなどの場合 0 以外の正の数を返します。
FM 音源が使用不可の場合、−1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

プログラム例

```
#include        <string.h>
#include        <basic0.h>
#include        <music.h>

#define NASI            'NASI'

static char     as[256 + 1];


main()
{
        int     i;

        b_init();
        m_init();
        for (i = 1; i <= 6; i++) {
                m_alloc(i, 2000);
                m_assign(i, i);
        }
        m_trk(1, "Q7 @34 V9   O4 T66"); /* trumpet */
        m_trk(2, "Q7 @35 V10  O4 T66"); /* horn */
        m_trk(3, "Q7 @37 V10  O3 T66"); /* tuba */
        m_trk(4, "Q8 @6  V8   O5 T66"); /* cembalo */
        m_trk(5, "Q8 @6  V8   O5 T66"); /* cembalo */
        m_trk(6, "Q8 @6  V8   O4 T66"); /* cembalo */

        /* Part 1 */
        strcpy(as, "L16 B-4&B-B-AGF8G8DE-F8 : ");
        strcat(as, "<R8F8&FE-DC>B-8<DA>AB-<C8");
        strcat(as, ">F<FD>B-<E-8G8&GE-C>A&A8<F8 : ");
        strcat(as, "D4E4F8>B-4A8");
        m_trk(1, as);
        m_trk(4, as);

        /* Part 2 */
        strcpy(as, "L16R1 : ");
        strcat(as, ">B-FB-<DC8E-8&E-8DE {FEFEFEFE}8 &ED32E
        strcat(as, "F4G8B-8E-8G8<C8C8& : ");
        strcat(as, "C>FB-<D>G8B-8A8DE-FGE-F");
        m_trk(2, as);
        m_trk(5, as);

        /* Part 3 */
        strcpy(as, "L8 B-<DC>B-AGFE- : ");
        strcat(as, "D4F4G4<C4:");
        strcat(as, "D4&L16DD>B-G<C8E-8&E-C>AF : ");
        strcat(as, "B-4&B-B-AGF8G8DE-F8");
        m_trk(3, as);
        m_trk(6, as);

        m_play(NASI, NASI, NASI, NASI, NASI, NASI, NASI, N.
        b_exit(0);
}
```

msarea

レベル３

書　式

```
#include <mouse.h>

int msarea (x1, y1, x2, y2) ;
int x1 ; /* マウスカーソルの移動範囲の始点X座標 */
int y1 ; /* マウスカーソルの移動範囲の始点Y座標 */
int x2 ; /* マウスカーソルの移動範囲の終点X座標 */
int y2 ; /* マウスカーソルの移動範囲の終点Y座標 */
```

機　能

マウスカーソルの移動範囲を設定します。
x1 にはマウスカーソルの移動範囲の始点X座標を指定します。
y1 にはマウスカーソルの移動範囲の始点Y座標を指定します。
x2 にはマウスカーソルの移動範囲の終点X座標を指定します。
y2 にはマウスカーソルの移動範囲の終点Y座標を指定します。
かならず始点が左上、終点が右下でなければなりません。
また、それぞれの指定可能範囲は以下の通りです。

$0 \leqq x1 \leqq 1022$
$0 \leqq y1 \leqq 1006$
$x1 < x2 \leqq 1023$
$y1 < y2 \leqq 1007$

戻り値

正常終了のとき、0 を返します。
エラーのとき、−1 を返します。

プログラム例

```
#include        <stdio.h>
#include        <conio.h>
#include        <basic0.h>
#include        <mouse.h>
#include        <graph.h>


main()
{
        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        mouse(0);
        msarea(0, 0, 500, 400);
        box(0, 0, 500, 400, 15, 0xffff);
        mouse(1);
        puts("move mousr cursor! (0, 0) <---> (500, 400)");
        while (kbhit() == 0)
              ;
        b_exit(0);
}
```

msbtn

レベル３

書　式　# include <mouse. h>

```
int msbth(n, b, t) ;
int n;  /*  マウスボタンの状態の指定  */
int b;  /*  マウスボタンの指定  */
int t;  /*  待ち時間  */
```

機　能　マウスボタンが、指定した状態になるまでの時間を通知します。
nにはマウスボタンの状態を指定します。
0を指定するとマウスボタンを離すまで、1を指定するとマウスボタンを押すまでの時間が測定されます。
bにはマウスボタンの左右の指定を行います。
0で左ボタン、1で右ボタンとなります。
tには待ち時間の最大値を指定します。
0～65535までを指定します。
0を指定すると、指定された状態が発生するまで待ち続けます。

戻り値　待ち時間を返します（1～65534）。
ドラッグが行われた場合は0、待ち時間を超えた場合は−1を返します。
引数のエラーの場合は−2を返します。

プログラム例

```
#include        <stdio.h>
#include        <basic0.h>
#include        <mouse.h>
#include        <graph.h>


main()
{
        int     d_clk(void);
        int     i, j;
        int     ni, nj;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        mouse(0);
        msarea(0, 0, 767, 511);
        box(0, 0, 767, 511, 15, 0xffff);
        mouse(1);
        puts("move line by double click!");
        for (i = j = 0; d_clk() != 1; i = ni, j = nj) {
                line(i, 0, i, 511, 0, 0xffff);
                line(0, j, 767, j, 0, 0xffff);
                box(0, 0, 767, 511, 15, 0xffff);
                mspos(&ni, &nj);
                line(ni, 0, ni, 511, 12, 0xffff);
                line(0, nj, 767, nj, 12, 0xffff);
        }
        b_exit(0);
}


static int      d_clk(void)
{
        int     x, y, bl, br;
        int     f;

        msstat(&x, &y, &bl, &br);
        if (bl != -1)
                return (0);
        f = msbtn(0, 0, 80);
        if (10 > f || f > 70)
                return (0);
        f = msbtn(1, 0, 80);
        if (10 > f || f > 70)
                return (0);
        f = msbtn(0, 0, 80);
        if (10 > f || f > 70)
                return (0);
        return (1);
}
```

mspos

レベル 3

書　　式　#include <mouse. h>

int mspos(x, y) ;
int　*x ; /*　X 座標を格納する変数を指すポインタ　*/
int　*y ; /*　Y 座標を格納する変数を指すポインタ　*/

機　　能　マウスカーソルの座標を変数に代入します。
x にはマウスカーソルの X 座標を代入する変数を指すポインタを指定します。
y には Y 座標を代入する変数を指すポインタを指定します。

戻 り 値　上位 16 ビットに X 座標、下位 16 ビットに Y 座標が返ります。

プログラム例

```
#include        <stdio.h>
#include        <conio.h>
#include        <basic0.h>
#include        <mouse.h>
#include        <graph.h>


main()
{
        int     x, y;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        mouse(0);
        msarea(0, 0, 767, 511);
        box(0, 0, 767, 511, 15, 0xffff);
        mouse(1);
        locate(10, 10);
        puts("move mouse cursor!(0, 0) <--->(767, 511)");
        while (kbhit() == 0) {
                mspos(&x, &y);
                locate(10, 11);
                printf("mouse cursor (X) = %3d¥n", x);
                locate(10, 12);
                printf("mouse cursor (Y) = %3d¥n", y);
        }
        b_exit(0);
}
```

msstat

レベル3

書　式

```
#include <mouse.h>

int msstat(x, y, bl, br) ;
int  *x;  /*  X方向の相対移動量を格納する変数を指すポインタ  */
int  *y;  /*  Y方向の相対移動量を格納する変数を指すポインタ  */
int  *bl; /*  左ボタンの状態を格納する変数を指すポインタ  */
int  *br; /*  右ボタンの状態を格納する変数を指すポインタ  */
```

機　能

マウスの状態を変数に代入します。
xにはX方向の相対移動量を代入する変数を指すポインタを指定します。
−128～127が代入されます。
yにはY方向の相対移動量を代入する変数を指すポインタを指定します。
−128～127が代入されます。
blには左ボタンの状態を代入する変数を指すポインタを指定します。
左ボタンを押していた場合には−1が、押していない場合には0が代入されます。
brには右ボタンの状態を代入する変数を指すポインタを指定します。
右ボタンを押していた場合には−1が、押していない場合には0が代入されます。

戻り値

上位から8ビットごとに、Xの移動量、Yの移動量、左ボタンの状態、右ボタンの状態が返ります。

プログラム例

```
#include        <conio.h>
#include        <basic0.h>
#include        <mouse.h>
#include        <graph.h>


main()
{
        int     x, y;
        int     bl, br;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        mouse(0);
        msarea(0, 0, 767, 511);
        box(0, 0, 767, 511, 15, 0xffff);
        mouse(1);
        locate(10, 10);
        puts("move mouse cursor!(0, 0) <--->(767, 511)");
        locate(10, 11);
        cputs("mouse cursor relative movement");
        locate(10, 14);
        cputs("mouse button status");
        while (kbhit() == 0) {
                msstat(&x, &y, &bl, &br);
                locate(18, 12);
                cprintf(" (X) = %4d", x);
                locate(18, 13);
                cprintf(" (Y) = %4d", y);
                locate(18, 15);
                cprintf(" (LEFT) = %4d", bl);
                locate(18, 16);
                cprintf(" (RIGHT) = %4d", br);
        }
        locate(0, 17);
        b_exit(0);
}
```

m_stat

レベル3

書　　式

```
#include <music.h>

int m_stat(channel);
int channel;    /* チャンネル番号 */
```

機　　能

FM音源またはMIDIのchannelで指定したチャンネルの演奏状態を調べます。
channelには1〜24を指定します。
0または'NASI'を指定すると、1〜24すべてのチャンネルの状態を調べます。

戻 り 値

正常終了の場合はチャンネルの状態を返します。
このとき、channelに指定した値により以下のようになります。

● channelが1〜8のとき（本体のFM音源）

0　：演奏中でない
1　：演奏中

● channelが9〜24のとき（MIDIボード出力）

0　：演奏中でない
1　：演奏中

ただし、演奏中であってもmd_offによって出力が停止している場合があります。
また、MIDIボードがない場合には、常に0になります。

● channelが0または'NASI'のとき

ビット0〜7　：1〜8チャンネルの状態
ビット9〜23　：9〜24チャンネルの状態
(MIDIボードがないとき、ビットは0になります)
ビット24〜31：常に0

ビットが0で演奏中でない、1で演奏中であることを表します。

FM音源が使用不可の場合、−1を返します。

m_stop　レベル３

書式

```
#include <music.h>

int m_stop(c1, c2, c3, c4, c5, c6, c7, c8) ;
int c1, c2, c3, c4, c5, c6, c7, c8 ;  /*  チャンネル  */
```

機能

FM 音源のチャンネルを指定して演奏を一時中断します。
指定するチャンネルの個数が 7 個以下の場合は、余りの引数に 'NASI' を指定しなければなりません。
全チャンネルが 'NASI' ならば、1〜8 チャンネル中、現在演奏していたチャンネルすべてを一時中断します。
チャンネルは、1〜8 が指定できます。

戻り値

正常終了の場合は 0 を返します。
引数のエラーなどの場合、0 以外の正の数を返します。
FM 音源が使用不可の場合、−1 を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

m_tempo

レベル 3

書　式

```
#include <music.h>

int m_tempo(tempo) ;
char tempt ;  /*  テンポ  */
```

機　能

FM 音源および MIDI の演奏時のテンポを指定します。
テンポは、1分間に4分音符をいくつ演奏できるかで指定します。
たとえば、120 を指定すると、4分音符 120 拍で1分になります。
テンポに指定できる範囲は、32～200 です。

割り込み処理や、ディスクアクセスなどによって、演奏速度に誤差が発生する場合があります。

戻り値

正常終了の場合は0を返します。
引数のエラーなどの場合、0以外の正の数を返します。
FM 音源が使用不可の場合、−1を返します。
なお、エラー番号とその意味は、m_alloc 関数を参照してください。

m_trk

レベル3

書式

```
#include <music.h>

int m_trk(t, st);
char t;  /*  トラック番号  */
char *st;  /*  MMLデータへのポインタ  */
```

機能

MMLによる楽譜データをtで指定したトラックに書き込みます。
MMLの書式については、本体添付の「BASICマニュアル」を参照してください。

戻り値

正常終了の場合は0を返します。
引数のエラーやMMLの書式誤りなどの場合、0以外の正の数を返します。
FM音源が使用不可の場合、−1を返します。
なお、エラー番号とその意味は、m_alloc関数を参照してください。

m_vget　レベル3

書　式

```
#include <music2.h>

int m_vget(vo, ca) ;
char vo ; /*  音色番号  */
char *ca ; /*  音色データを格納する配列へのポインタ  */
```

機　能

FM音源のvoで指定した音色番号の音色データをcaが指す配列に読み込みます。
voには音色番号を指定します(1～200。ただし69～200は登録されていません)。
caには、読み込んだ音色データを格納するための二次元配列へのポインタを指定します。
音色データの形式は次の通りです。

```
char ca [5] [11] ;
```

	0		1～4（オペレータ）	
0	フィードバック/アルゴリズム	(0～63)	AR	(0～31)
1	スロットマスク	(0～15)	D1R	(0～31)
2	ウェーブフォーム	(0～3)	D2R	(0～31)
3	シンクロ	(0, 1)	RR	(0～15)
4	スピード	(0～255)	D1L	(0～15)
5	PMD	(0～127)	TL	(0～127)
6	AMD	(0～127)	KS	(0～3)
7	PMS	(0～7)	MUL	(0～15)
8	AMS	(0～3)	DT1	(0～7)
9	L, RPAN	(0～3)	DT2	(0～3)
10	未使用		AMS Enable	(0, 1)

上記の表中、( ) 内の数値はそれぞれのデータの範囲を示します。

音色データの詳細については、本体添付の「BASICマニュアル」を参照してください。

戻り値

正常終了の場合は0を返します。
引数のエラーなどの場合0以外の正の数を返します。
FM音源が使用不可の場合、－1を返します。
なお、エラー番号とその意味は、m_alloc関数を参照してください。

m_vset

レベル3

書　式

```
#include <music2.h>

int m_vset(vo, ca);
char vo;   /*  音色番号  */
char *ca;  /*  音色データを格納してある配列へのポインタ  */
```

機　能

配列に格納してあるFM音源用の音色データを、音色に登録します。
voには音色番号を指定します（1～200。ただし1～68は登録済み）。
caには音色データを格納してある二次元配列へのポインタを指定します。
音色データの形式は次の通りです。

```
char ca[5][11];
```

	0		1～4（オペレータ）	
0	フィードバック/アルゴリズム	(0～63)	AR	(0～31)
1	スロットマスク	(0～15)	D1R	(0～31)
2	ウェーブフォーム	(0～3)	D2R	(0～31)
3	シンクロ	(0, 1)	RR	(0～15)
4	スピード	(0～255)	D1L	(0～15)
5	PMD	(0～127)	TL	(0～127)
6	AMD	(0～127)	KS	(0～3)
7	PMS	(0～7)	MUL	(0～15)
8	AMS	(0～3)	DT1	(0～7)
9	L, RPAN	(0～3)	DT2	(0～3)
10	未使用		AMS Enable	(0, 1)

上記の表中、（　）内の数値はそれぞれのデータの範囲を示します。

音色データの詳細については、本体添付の「BASICマニュアル」を参照してください。

戻り値

正常終了の場合は0を返します。
引数のエラーなどの場合0以外の正の数を返します。
FM音源が使用不可の場合、−1を返します。
なお、エラー番号とその意味は、m_alloc関数を参照してください。

nfiles

レベル2

書　　式

```
#include <io.h>

char   *nfiles(buff, atr, datetime, len) ;
char   *buff ;  /*  検索用バッファへのポインタ  */
char   *atr ;  /*  見つけたファイルの属性を格納する領域へのポインタ  */
int   *datetime ;  /*  見つけたファイルの作成日時を格納する領域へのポイ
                        ンタ  */
size_t   *len;  /*  見つけたファイルのバイト数を格納する領域へのポインタ  */
```

機　　能

ファイルの次の候補を検索し、見つけたファイルの情報をatr, datetime, lenで指定した領域に格納し、戻り値として見つけたファイルのファイル名へのポインタを返します。

buffは、files関数で指定した検索用バッファへのポインタです。

この関数を呼び出す前には、files関数を呼び出しておかなければなりません。

atrは、見つけたファイルの属性を格納する領域へのポインタです。

atrの指す領域のサイズは1バイト以上必要です。

＊artに格納された属性は、各ビットが1の場合に以下の意味を表しています。

ビット0：読み込み専用
ビット1：隠しファイル
ビット2：システムファイル
ビット3：ボリューム名
ビット4：ディレクトリ
ビット5：通常ファイル

datetimeは、見つけたファイルの作成日時を格納する領域へのポインタです。

datetimeの指す領域のサイズは4バイト以上必要です。

また、＊datetimeに格納される作成日時の形式は以下の通りです。

ビット31〜25　：1980年からの相対年数(0〜99)
ビット24〜21　：月(1〜12)
ビット20〜16　：日(1〜31)
ビット15〜11　：時(0〜23)
ビット10〜5　：分(0〜59)
ビット4〜0　：秒/2(0〜29)

len は、見つけたファイルのバイト数を格納する領域へのポインタです。
len の指す領域のサイズは 4 バイト以上必要です。
files 関数を実行した後で、nfiles をエラーが返されるまで実行することで、files 関数で指定しておいた条件を満たすファイル全てを検索することができます。

戻り値

正常終了ならば、見つけたファイルのファイル名へのポインタを返します。
なお、このファイル名は buff で指定した検索用バッファに作成されるため、同一の検索用バッファを指定して files 関数や、nfiles を呼び出すと以前の内容は失われることに注意してください。
エラーならば NULL を返します。

参照関数

chgfa, chgft, files

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <io.h>

#define A_RDO           0x01    /* 読み込み専用 */
#define A_HDN           0x02    /* 隠しファイル */
#define A_SYS           0x04    /* システムファイル */
#define A_VOL           0x08    /* Volumeラベル */
#define A_DIR           0x10    /* ディレクトリ */
#define A_FIL           0x20    /* 通常ファイル */

static char     *pattern = "¥¥usr¥¥include¥¥*.h";
static char     buf[150];
static char     atr;
static long     dt;
static size_t   len;


main()
{
        char    *name;

        name = files(buf, &atr, &dt, &len, pattern, A_FIL);
        if (name != NULL)
                printf("%s¥n", name);
        else
                exit(1);

        while ((name = nfiles(buf, &atr, &dt, &len)) != NULL)
                printf("%s¥n", name);

        exit(0);
}
```

offsetof

レベル2

書式

```
#include <stddef.h>

size_t offsetof (tag, rzrber) ;
(type)tag;
(identifier)member;
```

機能

offsetofは、tagの構造体での構造体メンバmemberの位置を求めるマクロです。
tagには、構造体名を指定します。
memberには、メンバ名を指定します。
ビットフィールドの構造体は指定できません。

戻り値

tagの構造体での、構造体メンバmemberの位置を返します。

open

レベル2

書　式

```
#include 〈fcntl. h〉
#include 〈stat. h〉
#include 〈io. h〉

int open(pathname, omode [, pmode] ) ;
char   *pathname; /*  ファイル名へのポインタ  */
int omode; /*  モード  */
int pmode; /*  アクセスモード  */
```

機　能

pathnameで指定したファイルをomodeで指定したモードでオープンします。
omodeは下記に示す定数の組み合わせで指定します。
複数個の定数を組み合わせるときは、OR'|'でつなげます。
例:O_TEXT | O_RDWR

omodeの値	内　容
O_APPEND	ファイルポインタを書き込み操作の前にエンドオブファイルに移動する
O_CREAT	書き込みのために新しいファイルを作る ただし、ファイルが存在していればこのモードは無効となる
O_EXCL	ファイルが存在した場合にエラー値を返す O_CREATとともに指定したときのみ有効となる
O_RDONLY	読み出し専用としてファイルをオープンする O_RDWR、O_WRONLYを同時に指定できない
O_RDWR	読み書きのためにファイルをオープンする O_RDONLY、O_WRONLYを同時に指定できない
O_TRUNC	ファイルが存在していれば、ファイルの内容を空にする O_RDWRまたはO_WRONLYとともに指定したときのみ有効となる
O_WRONLY	書き込み専用としてファイルをオープンする O_RDONLY、O_RDWRを同時に指定できない
O_BINARY	ファイルをバイナリモードでオープンする
O_TEXT	ファイルをテキストモードでオープンする

O_TRUNC は、既存のファイルの内容をすべて切り捨ててしまいます。
慎重に使用してください。

omode に O_CREAT を指定したときは、必ず pmode を指定します。
ただしファイルがすでに存在している場合は、pmode は無視されます。
pmode の値は chmod 関数での指定と同じです。

戻り値

オープンされたファイルのファイルハンドルを返します。
エラーが発生したときは−1を返し、errno に下記の値がセットされます。

errno の値	内容
EACCES	指定したファイル名がディレクトリである またはアクセスモードが違っている
EEXIST	O_CREAT、O_EXCL を指定したが、ファイルがすでに存在している
EMFILE	オープンするファイルの数が多すぎる
ENOENT	指定したファイル名またはパス名が見つからない

参照関数

access, chmod, close, creat, dup, dup2, fopen, sopen, umask

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <io.h>
#include        <stat.h>


main(int argc, char *argv[])
{
        char    *file;
        int     fn;

        file = (argc > 1) ? argv[1] : "file";

        if ((fn = open(file, O_RDONLY)) == -1) {
                printf("%sをopenできません。¥n", file);
                exit(1);
        }
        printf("%sをopenしました。¥n", file);

        if (close(fn) == -1) {
                printf("%sをcloseできません。¥n", file);
                exit(1);
        }
        printf("%sをcloseしました。¥n", file);

        exit(0);
}
```

paint

レベル3

書　式

#include <graph. h>

void paint(x, y, p) ;
int x ; /*　X座標　*/
int y ; /*　Y座標　*/
int p ; /*　パレットコード　*/

機　能

グラフィック画面のx、yで指定された単色領域をpで塗りつぶします。
xには領域内の1点のX座標を指定します（−32768〜32767）。
yには領域内の1点のY座標を指定します（−32768〜32767）。
pにはパレットコードを指定します（0〜65535）。
画面モードにより最大値が異なります。

戻り値

戻り値はありません。

参照関数

screen

プログラム例

```
#include        <stdlib.h>
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i;
        int     b;
        int     c;

        b_init();
        screen(1, 3, 1, 1);
        vpage(1);
        for (i = 0; i < 500; i++) {
                b = rand() * 2 + 1;
                c = rand() * 2 + 1;
                circle(255, 255, 100, b, 0, 360, 256);
                paint(255, 255, c);
        }
        b_exit(0);
}
```

palet

レベル 3

書　　式　#include <graph. h>

void palet(p, c) ;
int p; /＊ パレットコード ＊/
int c; /＊ 色コード ＊/

機　　能　パレットに色コードを割りあてます。
pにはパレットコードを指定します (0～65535)。
画面モードにより最大値が異なります。
cには色コードを指定します (0～65535)。
なお、65536色モードでは、1つの色を変化させると、同時に関連する256色が変化しますので、原則的に色コードとパレットコードは同じ値を指定します。

戻 り 値　戻り値はありません。

参照関数　screen

プログラム例

```
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     a, b, c, d, e;
        int     f;

        b_init();
        screen(1, 3, 1, 1);
        vpage(1);
        for (a = 0; a < 2; a++) {
                for (b = 31; b >= 0; b--) {
                        for (c = 31; c >= 0; c--) {
                                for (d = 0; d < 192; d++) {
                                        f = hsv(d, c, b) + 1 - a;
                                        palet(15, f);
                                        for (e = 0; e < 300; e++)
                                                ;
                                }
                        }
                }
        }
        b_exit(0);
}
```

perror

レベル2

書式

#include <stdio. h>

void perror(string) ;
char　*string ;　/*　ユーザー定義エラーメッセージへのポインタ　*/

機能

標準エラー出力（stderr）に対してエラーメッセージを出力します。
エラーメッセージはstring、コロン（:）、そしてエラーを起したライブラリに対するシステムエラーメッセージからなります。
stringに空文字列かNULLを指定すると、システムエラーメッセージのみを出力します。エラー番号は、グローバル変数errnoが使用されます。
もし、errnoが0以上の値ならば、perrorは何もしません。
システムエラーメッセージは、errnoを使ってエラー番号順に並んだ文字列のポインタ配列sys_errlistから探し出されます。
変数sys_nerrの値は、sys_errlist配列の要素数です。
errnoの値は何らかの関数の呼び出し時におけるエラーの発生によって変化します。
この関数で参照されるグローバル変数には、以下のものがあります。

int errno ;　/*　エラー番号　*/
int sys_nerr ;　/*　sys_errlistの配列要素数　*/
char　*sys_errlist [] ;　/*　エラーメッセージのポインタ配列　*/

戻り値

戻り値はありません。

参照関数

strerror

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <process.h>


main(int argc, char *argv[])
{
        char    *name;
        FILE    *fp;

        name = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(name, "r")) == NULL) {
                perror(name);
                abort();
        }
        (void)fclose(fp);
        exit(0);
}
```

pi

レベル３

書　　式　#include <basic0.h>

```
double pi( ) ;
```

機　　能　円周率 π の値を求めます。

戻 り 値　π の値を返します。

参照関数　b_pi

point

レベル3

書　式

#include <graph. h>

```
int point(x, y) ;
int x ; /*  X座標  */
int y ; /*  Y座標  */
```

機　能

グラフィック画面上の指定座標のドットのパレットコードを調べます。
x には調べる点の X 座標を指定します（−32768～32767)。
y には調べる点の Y 座標を指定します（−32768～32767)。

戻り値

指定座標のドットのパレットコードを返します。
ただし、window 関数によって設定されたクリッピング領域の範囲外の座標を指定した場合は、たとえ指定座標にドットがあったとしても、常に0を返します。

参照関数

pset, window

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i;
        int     b;
        int     c;
        int     d;

        b_init();
        screen(1, 3, 1, 1);
        vpage(1);
        for (i = 0; i < 500; i++) {
                b = rand() * 2 + 1;
                c = rand() * 2 + 1;
                circle(255, 255, 100, b, 0, 360, 256);
                paint(255, 255, c);
                d = point(255, 255);
                printf("palet code = %d¥n", d);
        }
        b_exit(0);
}
```

pos

レベル 3

書　　式　# include <basic0. h>

```
int pos( ) ;
```

機　　能　カーソルの現在位置のX座標を得ます。

戻 り 値　カーソルのX座標を返します。

参照関数　csrlin

pow

レベル2

書　　式

```
#include <math.h>

double pow(x, y) ;
double     x ;
double     y ;
```

機　　能

x の y 乗を計算します。

戻 り 値

x^y の値を返します。
x が 0 で y が 0 か負の数の場合、または x が負の数で y が整数でない場合の、戻り値は不定で、errno に EDOM がセットされます。

参照関数

exp, log, sqrt

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        double  y;
        int     i;
        int     j;

        for (i = -32; i <= 32; i++) {
                for (j = -32; j <= 32; j++) {
                        x = (double)i;
                        y = (double)j;
                        printf("pow(%+4.1f, %+4.1f) = %g¥n",
                         x, y, pow(x, y));
                }
        }
}
```

printf

レベル 2

書　式

```
#include <stdio.h>

int printf(format_string [, arg. . . ] ) ;
char  *format_string ;  /*  書式文字列へのポインタ  */
```

機　能

format_string で指定した書式に従って出力を変換し、標準出力 stdout に出力します。

format_string に続く引数は、書式に従って変換され出力されます。

format_string の中のフォーマット指定の出現順、およびそれが要求する型と、引数 arg の出現順およびその型は一致していなければなりません。

フォーマット指定の書式を以下に示します。

% [flag] [length] [. digit] [lentype] type

"%" と type 以外は省略できます。

フォーマット指定の各フィールドの内容は次の通りです。

フィールド	内　　容
%	書式指定の始まりを示す 省略できない
flag	符号、空白、小数点、8 進、16 進などの出力の修飾を指定する
length	出力文字の最小文字数を指示する
digit	最小桁数、または小数点以下の桁数を指定する
lentype	引数の型を指定する
type	変換のタイプを指定する

"%"のうしろに format_string として無意味な文字があると、そのまま出力します。

各フィールドの内容は次の通りです。

【flag】

flag の値と内容は次の通りです。

flag	内　　容
−	length の桁数内で左揃えで出力する 省略すると右揃えで出力する
＋	出力数値が正の数のときは“＋”を、負の数のときは“−”をつけて出力する 省略すると、負の数のときのみ“−”をつけ、正の数のときは符号をつけない
空白	出力数値が正の数のときは1文字分の空白を、負の数のときは“−”をつける 省略すると、負の数のときのみ“−”をつけ、正の数のときは空白をつけない flag に“＋”が指定されているときは、この指定は無効になる
#	type の値が、0、x、X、b、B のとき、出力数値の先頭に 0、0x、0X、0b、0B をつける 省略すると何もつけない
	type の値が f、e、E のとき、小数点以下に数字がなくても小数点を付ける 省略すると、小数点以下に数字があるときのみ小数点を付ける
	type の値が g、G のとき、小数点以下の数字が digit で指定した桁数に足りなくても、切り捨てや余分な 0 の削除は行わない 省略すると、小数点以下に数字があるときのみ小数点以下か付いて、余分な 0 は削除する
	type の値が c、d、u、s のときは無視される

【length】

出力する最小文字数を 10 進数の整数で指定します。

出力する文字数が length よりも少ない場合は、文字の左側を空白で埋めて右揃え（マイナスフラグが指定されているときは逆）で出力されます。

出力する文字数が length よりも多い場合、または length が指定されていない場合は、すべての文字を出力します。

lengthを引数で指定する場合には、アスタリスク（*）を使用します。
lengthに対応する引数は、フォーマット出力する引数の前になければならず、int型でなければなりません。

【digit】

digitは出力する最大文字数または小数点以下の桁数を正の10進数の整数で指定します。
digitを引数で指定する場合には、アスタリスク（*）を使用します。
digitに対応する引数は、フォーマット出力する引数の前になければならず、int型でなければなりません。
digitの指定方法、および省略した場合のデフォルト値は次の通りです。

typeの値	内　　容
d u b B o x X	出力する最小桁を指定する 桁数がdigitより少ない場合は0を付加する 桁数がdigitを超えても値の切り捨ては行われず、そのまま出力する 省略したとき、0を指定されたときは“1”とみなす
f e E	小数点以下の桁数を指定する 出力する最後の桁は四捨五入される 省略したときは6桁になる 0を指定したとき、またはピリオドのみが記述されたときは、小数点以下の数字を切り捨てる
g G	出力する有効値の最大桁数を指定する 省略したときは有効値のすべてを出力する
c	無効
s	出力する最大文字数を指定する 省略したときは、文字列の末尾のヌル文字に達するまで出力する

【lentype】

lentypeは出力する引数の型を指定します。

lentypeの内容は次の通りです。

lentypeの値	内　　容
l L	typeの値がd、u、b、o、xのとき、引数がlongかunsigned longであることを指定する
h H	typeの値がd、u、b、o、xのとき、引数がshortかunsigned shortであることを指定する

【type】

typeの値	引　数	内　　容
d i	整数型	符号つき10進整数
u	整数型	符号なしの10進整数
b	整数型	符号なしの2進整数
B	整数型	符号なしの2進整数
o	整数型	符号なしの8進整数
x	整数型	符号なしの“abcdef”を使った16進整数
X	整数型	符号なしの“ABCDEF”を使った16進整数
f	double型	double値を［−］ddd.dddの小数点形式に変換して出力する 小数点以下の桁数はdigitにより指定されます
e	double型	double値を［−］d.ddd e［−］dddの指数形式に変換して出力する 小数点以下の桁数はdigitにより指定される
E	double型	指数の前に“e”の代わりに“E”が使われる以外はeフォーマットと同じ

typeの値	引　数	内　容
g	double型	double値を、fフォーマット（小数点形式）またはeフォーマット（指数形式）のうち、結果が短くなる形式に変換して出力する 指数部が−4以下または桁数がdigitよりも大きくなる場合はeフォーマットになる
G	double型	指数の前に“e”の代わりに“E”が使われる以外はgフォーマットと同じ
c	文　字	1文字
s	文字列	ヌル文字（'¥0'）で終了する文字列 またはdigitの数の文字列
p	ポインタ型	ポインタのアドレスを16進数出力
n	int＊型	現在までに出力した文字数を引数に代入する 出力は行なわない

戻り値　出力した文字数を返します。

参照関数　fprintf, scanf, sprintf

プログラム例

```
#include        <stdio.h>


main()
{
        printf("hello world¥n");
        printf("integer[%d]¥n", 200);
        printf("integer[%x]¥n", 200);
        printf("float[%f]¥n", 3.1415926);
        printf("float[%g]¥n", 3.1415926);
        printf("character[%c]¥n", 'A');
        printf("string[%s]¥n", "string");
}
```

pset

レベル3

書　式

#include <graph. h>

```
void pset(x, y, p) ;
int x ; /*  X座標  */
int y ; /*  Y座標  */
int p ; /*  パレットコード  */
```

機　能

グラフィック画面に点（ドット）を描画します。
xには点を描くX座標を指定します（−32768〜32767）。
yには点を描くY座標を指定します（−32768〜32767）。
pにはパレットコードを指定します（0〜65535）。
画面モードにより最大値が異なります。

戻り値

戻り値はありません。

参照関数

screen, point

プログラム例

```
#include        <basic0.h>
#include        <stdlib.h>
#include        <graph.h>

main()
{
        int     i;
        int     x;
        int     y;
        int     c;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        for (i = 0; i < 5000; i++) {
                x = rand() % 768;
                y = rand() % 200;
                c = rand() % 16;
                pset(x, y, c);
        }
        b_exit(0);
}
```

put

レベル3

書式

```
#include <graph.h>

void put(x1, y1, x2, y2, buf, size) ;
int x1 ; /*  始点X座標  */
int y1 ; /*  始点Y座標  */
int x2 ; /*  終点X座標  */
int y2 ; /*  終点Y座標  */
void  *buf ; /*  画像データ格納配列へのポインタ  */
int size ; /*  画像データのバイト数  */
```

機能

bufが示す配列に格納されたsizeバイトの画像データを、グラフィック画面の長方形の領域に描画します。

x1には始点X座標を指定します。
y1には始点Y座標を指定します。
x2には終点X座標を指定します。
y2には終点Y座標を指定します。

始点と終点はwindowで指定したクリッピング座標の中でなければいけません。
bufには、描画する画像データが格納してある領域へのポインタを指定します。
この領域には、sizeバイト分画像データが格納されていなければなりません。
sizには、画像データのバイト数を指定します。
画面モードにより、以下のバイト数を指定します。

16色モード：(総ドット数+1) ÷2
256色モード：総ドット数
65536色モード：総ドット数×2

戻り値

戻り値はありません。

参照関数

get, window

プログラム例

```
#include        <basic0.h>
#include        <graph.h>


main()
{
        static char     buf[14015 + 1];

        screen(1, 3, 1, 1);
        vpage(1);
        symbol(3, 1, "x68000", 2, 2, 2, 65535, 0);
        box(0, 0, 147, 49, 65535, 0xffff);

        get(1, 1, 146, 48, buf, sizeof(buf));

        put(200, 200, 345, 247, buf, sizeof(buf));
}
```

putc, putchar

レベル 2

書　　式

```
#include <stdio.h>

int putc(c, stream) ;
int c; /* 出力する文字 */
FILE *stream; /* FILE 構造体へのポインタ */

int putchar(c) ;
int c; /* 出力する文字 */
```

機　　能

putc は stream で指定するストリームに1文字出力します。
putchar は putc(c, stdout) と同じになります。

戻 り 値

出力した文字を返します。
エラーが発生した場合は、EOF を返します。

参照関数

fputc, fputchar, getc, getchar

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main(int argc, char *argv[])
{
        FILE    *fp;
        int     c;

        if (argc == 1) {
                while ((c = getchar()) != EOF)
                        putchar(c);
        }
        else {
                if ((fp = fopen(argv[1], "w")) == NULL) {
                        perror(argv[1]);
                        exit(1);
                }
                while ((c = getchar()) != EOF)
                        putc(c, fp);
                (void)fclose(fp);
        }
        exit(0);
}
```

注：putc、putchar は fputc、fputchar 関数と同じですが、マクロ定義と関数の両方があります。

マクロ定義で使いたい場合は、stdio. h をインクルードする前に次の1行を追加してください。

```
# define MACRO
```

putch

レベル２

書　　式

#include <conio.h>

int putch(c) ;
int c ; /＊　コンソールに出力する文字　＊/

機　　能

コンソールに文字 c を出力します。
改行文字'¥n'を、復帰改行に変換しません。
正しく復帰改行したい場合は、'¥r'と'¥n'を続けて出力してください。

戻 り 値

成功した場合は出力した文字 c を返します。失敗した場合は EOF を返します。

参照関数

cprintf, getch, getche

プログラム例

```
#include        <conio.h>


main()
{
        int     c;

        c = getch();
        putch(c);
}
```

putenv　レベル2

書　　式

```
#include <stdlib.h>

int putenv(envstring);
char  *envstring;  /*  環境文字列へのポインタ  */
```

機　　能

新しい環境変数の追加、および既存の環境変数の修正、削除を行います。
envstring は次のような形式の文字列へのポインタです。

```
varname = string
```

varname は追加、または修正する環境変数の名前で、string は環境変数の値です。
varname がすでに環境変数として存在する場合には、その string を指定した string によって置き換え、そうでなければ新しい環境変数として追加します。
空の string を指定する（"varname="を指定）すると、varname を環境から削除します。

戻 り 値

正しく追加、修正削除できた場合には、0を返します。
エラーが発生した場合－1を返します。

参照関数

getenv

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define NAME            "path"

static char     buf[256];


main()
{
        char    *p;

        if ((p = getenv(NAME)) == NULL) {
                printf("%s is not found.¥n", NAME);
                exit(1);
        }
        else
                printf("%s = %s¥n", NAME, p);

        sprintf(buf, "%s=%s;¥¥usr¥¥games", NAME, p);
        if (putenv(buf)) {
                printf("can't put env.¥n");
                exit(1);
        }
        p = getenv(NAME);
        printf("%s = %s¥n", NAME, p);
        exit(0);
}
```

putl

レベル２

書　式

```
#include <stdio.h>

int putl(lg, fp);

long lg;  /*  書き出す値  */
FILE  *fp;  /*  ファイル構造体へのポインタ  */
```

機　能

fpで指定されたストリームにlongサイズ分書き出します。

戻り値

書き出したlongの値を返します。
エラーが発生した場合はEOFを返します。
ただし、ストリームに書き込んだデータがEOFと一致している可能性がありますので、エラーの判定はferror関数を使用して判定してください。

参照関数

getw, putw, getl

puts

レベル2

書式

#include <stdio. h>

int puts(string) ;
char *string; /* 出力する文字列へのポインタ */

機能

指定したstringを、標準出力stdoutへ出力します。
このとき文字列の終了を示すヌル文字'¥0'は改行文字'¥n'に置き換えられます。

戻り値

成功すると0を、失敗するとEOFを返します。

参照関数

fputs, gets

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define SIZE    4096

static char     buf[SIZE];


main()
{

        while (gets(buf) != NULL)
                puts(buf);
        exit(0);
}
```

putw

レベル２

書　式

```
#include <stdio.h>

int putw(sh, stream) ;
short sh ;  /*  書き出す値  */
FILE  *stream ;  /*  FILE構造体へのポインタ  */
```

機　能

streamで指定されたストリームにshortサイズ分書き出します。

戻り値

書き出したshortの値を返します。
エラーが発生した場合は、EOFを返します。

参照関数

getw, getl, putl

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        short   s;
        FILE    *fp;

        if ((s = getw(stdin)) == EOF) {
                if (ferror(stdin)) {
                        perror("fail");
                        clearerr(stdin);
                }
                else if (eof(stdin))
                        printf("end of file¥n");
                exit(1);
        }
        fprintf(stderr, "number = %d¥n", s);
        putw(s, stdout);
        exit(0);
}
```

qsort

レベル 2

書　式　#include <stdlib. h>

```
void qsort(base, num, width, compare) ;
void  *base;  /*  配列を指すポインタ  */
size_t num, width ;  /*  配列要素の数、配列要素のサイズ  */
int (*compare) (const void *, const void *) ; /*
compare ルーチンを指すポインタ  */
```

機　能

各要素が width バイトで、要素数が num 個の配列をクイックソートします。
base はソートする配列を指すポインタです。
compare は配列の 2 つの要素を比較し、その大小関係を返すユーザーが用意するルーチンを指すポインタです。
qsort はソートの過程で、必要に応じて compare が指す関数を呼び出します。
compare は 2 つの引数を受けとります。引数には配列の要素へのポインタが渡されますので、その 2 つを比較して、次の値のうちいずれかを返すようにします。

compare の戻り値	内　　容
負	要素 1 は要素 2 より小さい
0	要素 1 と要素 2 は等しい
正	要素 1 は要素 2 より大きい

戻り値

戻り値はありません。

参照関数　bsearch

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>

#define MAX     12


main(int argc, char *argv[])
{
        int     compare(char **, char **);

        qsort(argv, argc, sizeof(char *), compare);
        while (argc--)
                printf("%s¥n", *argv++);
}


static int      compare(char **key1, char **key2)
{
        return (strcmp(*key1, *key2));
}
```

raise

レベル２

書　式

```
#include <signal.h>

int raise(sig);
int sig;  /*  送りたいシグナル  */
```

機　能

プログラムに、sigで指定されたシグナルを送ります。
sigはsignal. hで定義されたマクロSIGINT, SIGABRT, SIGTERMのいずれかを指定します。sigの意味を以下に示します。

sigの値	意　味
SIGINT	CTRL+C、BREAKキーによる割り込み
SIGABRT	異常終了
SIGTERM	プログラム終了の要求

signal関数によって、各シグナルの処理が変更されていない場合かまたは、SIG_DFLが指定されていた場合は、raiseによって次の処理が実行されます。
SIGINT型シグナルを送ると、CTRL+C、BREAKキーを押した場合に引き起こされる処理が実行されます。
SIGABRT型シグナルを送ると、abort関数を呼び出したときに引き起こされる処理と同等な処理が実行されます。
signal関数によって、SIG_IGNが指定されていた場合は、raiseはエラーを返します。
signal関数によって、なんらかの関数が指定されていた場合は、raiseの呼び出しに引き続いてその関数が呼び出されます。
呼び出された関数には、引数として、raiseに指定されたシグナルsigがそのまま受けわたされます。
呼び出された関数は、そのシグナルにみあった処理を実行後に、signal (sig, SIG_DFL) を行い、続いてraise (sig) を行い、デフォルト処理に移行してください。
原則として、関数からリターンしてはいけません。
もし、関数からリターンしなければならない場合、速やかに0以外の値を戻り値としてリターンしてください。
raise関数は、その値を受け取って呼び出し元へ返します。
これにより、raiseを、見かけ上、失敗させることができます。

戻り値　成功すると0を、失敗すると0以外を返します。

参照関数　signal, abort

プログラム例

```
#include        <stdio.h>
#include        <signal.h>
#include        <stdlib.h>


int     func(int sig)
{
        int     res;

        res = 1;
        switch (sig) {
        case SIGINT:
                printf("SIGINTシグナルが発生しました¥n");
                break;
        case SIGABRT:
                printf("SIGABRTシグナルが発生しました¥n");
                break;
        case SIGTERM:
                printf("SIGTERMシグナルが発生しました¥n");
                break;
        default:
                res = -1;
                break;
        }
        return (res);
}


main()
{
        signal(SIGINT, (void(*)())func);
        signal(SIGABRT, (void(*)())func);
        signal(SIGTERM, (void(*)())func);
        printf("raise(SIGINT) の戻り値 %d¥n", raise(SIGINT));
        printf("raise(SIGABRT)の戻り値 %d¥n", raise(SIGABRT));
        printf("raise(SIGTERM)の戻り値 %d¥n", raise(SIGTERM));
}
```

rand

レベル 2

書　　式　# include <stdlib. h>

int rand( ) ;

機　　能　0～RAND_MAX の範囲の疑似乱数を発生します。
マクロ RAND_MAX は stdlib. h に定義されています。
srand 関数で乱数系列を初期化できます。

戻 り 値　0～RAND_MAX の範囲の擬似乱数を返します。

参照関数　srand

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define MAX     128


main(int argc, char *argv[])
{
        int     i;
        int     r;

        if (argc == 1)
                r = 0;
        else
                r = atoi(argv[1]);

        if (r == 0) {
                for (i = 0; i < MAX; i++)
                        printf("%10d¥n", rand());
        }
        else {
                for (i = 0; i < MAX; i++)
                        printf("%10d¥n", rand() % r);
        }
}
```

randomize

レベル3

書　式

```
#include <basic.h>

void randomize(i) ;
int i; /*  初期値  */
```

機　能　rnd関数の乱数系列の初期化を行います。

戻り値　戻り値はありません。

rbrk

レベル２

書　式

```
#include <stdlib.h>

void rbrk();
```

機　能

ヒープ領域のブレーク値（領域の上限値）を初期状態に戻します。
ヒープ領域とは、malloc 関数などが返すメモリ領域の全体をいいます。

戻り値

戻り値はありません。

注：malloc 関数などによって割りつけられたメモリ領域を解放せずに、本関数を実行してはいけません。

read　レベル2

書　式

```
#include <io.h>

size_t read(file, buffer, length);
int file;  /* 読み込むファイルのファイルハンドル */
void *buffer;  /* 読み込み領域を指すポインタ */
size_t length;  /* 読み込みバイト数 */
```

機　能

fileで指定されたファイルハンドルからlengthバイトを読み込み、bufferが指す領域に格納します。
読み込み操作は、指定したファイルの現在位置から始まります。
ファイルポインタは読み込み後には、読み込んだ文字の次に進みます。

戻り値

実際に読み込んだバイト数を返します。
ファイルにlengthバイトより少ないバイトしか残っていなかった場合は、lengthよりも小さい値になります。
エンドオブファイル（EOF）を読み込んだ場合は0を返します。
またエラーが発生した場合は−1を返し、errnoにEBADFがセットされます。

errnoの値	内　容
EBADF	指定したファイルが無効、またはファイルが読み出しのためにオープンされていない

ファイルをテキストモードでオープンした場合、戻り値は実際に読み込んだバイト数と等しくはなりません。
これはテキストモードでは、復帰改行'¥r¥n'を改行'¥n'に変換し、その変換した改行のみを戻り値のバイト数に加算するためです。
この変換は、ファイルポインタには影響しません。

参照関数

creat, fread, open, write

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <io.h>
#include        <stat.h>

#define SIZE    16


main(int argc, char *argv[])
{
        char    buf[SIZE];
        char    *file;
        int     fn;
        int     n;

        file = (argc > 1) ? argv[1] : "file";

        if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
                printf("can't open %s¥n", file);
                exit(1);
        }
        while (!eof(fn)) {
                n = read(fn, buf, SIZE);
                printf("%d bytes readed.¥n", n);
        }
        close(fn);
        exit(0);
}
```

realloc

レベル2

書式

```
#include <stdlib.h>

void    *realloc(ptr, size) ;
void    *ptr ;  /*  すでに割りつけられているメモリブロックへのポインタ  */
size_t size ;  /*  新しいメモリブロックのサイズ  */
```

機能

すでに割りつけられているメモリブロックの大きさを変更します。
ptr は malloc 関数や、realloc で返された、サイズを変更するメモリブロックのポインタを指定します。
size は新しいメモリブロックの大きさをバイト数で指定します。
大きさを変更してもメモリブロックの内容は変化しません。
ただし、新しいメモリブロックの位置は変化します。

戻り値

再割りつけしたメモリブロックを指すポインタを返します。
再割りつけられた領域は、どんな型のデータにも利用できるようになっています。
再割りつけ時にメモリブロックは移動するので、realloc の戻り値は必ずしも ptr と同じとはかぎりません。
メモリの不足などで指定した大きさのメモリブロックが確保できない場合には、NULL を返し、size が 0 であり ptr が NULL でないときはもとのメモリブロックが解放されます。

参照関数

calloc, free, malloc

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define B_SIZE          256


main()
{
        int     i;
        char    *p;
        char    *q;

        if ((p = malloc(B_SIZE)) == NULL) {
                printf("確保できませんでした。¥n");
                exit(1);
        }
        for (i = 0; i < B_SIZE; i++)
                p[i] = i;
        for (i = 1; i < 1024; i++) {
                printf("%4d block ", i);
                if ((q = realloc(p, (size_t)(i * B_SIZE))) == NULL) {
                        printf("再確保できませんでした。¥n");
                        break;
                }
                printf("確保しました。¥n");
                p = q;
        }
        for (i = 0; i < B_SIZE; i++)
                printf("p[%d] = %d¥n", i, p[i]);
        free(p);
        printf("解放しました。¥n");
        exit(0);
}
```

rename

レベル２

書式

```
#include <io.h>

int rename(oldname, newname) ;
char  *oldname ;  /*  旧ファイル名へのポインタ  */
char  *newname ;  /*  新ファイル名へのポインタ  */
```

機能

oldnameで指定したファイル名、またはディレクトリ名をnewnameで指定した名前に変更します。
oldnameは既存のファイルまたはディレクトリのパス名でなければなりません。
newnameに既存のファイル、またはディレクトリ名を指定することはできません。
newnameで異なったパス名を指定することにより、ファイルを他のディレクトリへ移すことができます。
ただし、異なったデバイスへ移動する（たとえば、ドライブAからドライブBへ）ことはできません。
また、ディレクトリは名前を変えることができるだけで、移動することはできません。
カレントディレクトリに係わる（先頭から一部、または全部に一致する）ディレクトリの名前を変えないでください。
もし変えると、その後の動作は保障しません。

戻り値

正常終了ならば正の数（0も）を返します。
エラーならば−1を返します。
また、エラーが発生した場合、errnoと_doserrnoにそれぞれCのエラー番号とOSのエラー番号をセットします。

参照関数

creat, fopen, open

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <io.h>


main(int argc, char *argv[])
{
        if (argc < 3) {
                puts("Usage: rename oldName newName");
                exit(1);
        }
        else {
                if (rename(argv[1], argv[2]))
                        puts("Rename error");
                else
                        puts("Success");
        }
        exit(0);
}
```

repmem

レベル２

書　式

```
#include <string.h>

void repmem(dest, source, size, n) ;
void  *dest ;  /*  コピー先領域のポインタ  */
void  *source ;  /*  コピー元領域のポインタ  */
size_t size ;  /*  コピーするバイト数  */
int n ;  /*  コピーする回数  */
```

機　能　source が指す領域の size バイトを、dest が指す領域に n 回コピーします。

戻り値　戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <string.h>


main()
{
        static char     dst[] = "abcdefghijklmnopqrstuvwxyz";
        static char     src[] = "x68000,";

        printf("%s¥n", dst);
        repmem(dst, src, strlen(src), 2);
        printf("%s¥n", dst);
}
```

rewind　　　レベル2

書　　式　#include <stdio.h>

void rewind(stream)；
FILE　＊stream；　/＊　FILE 構造体へのポインタ　＊/

機　　能　ファイル stream のポインタをファイルの先頭へ移動します。
fseek 関数でも同様の働きが可能ですが、fseek 関数がエンドオブファイル（EOF）やエラーインジケータをクリアしないのに対し、rewind はそれらをクリアします。

戻 り 値　戻り値はありません。

参照関数　fseek, ftell

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;

        file = (argc > 1) ? argv[1] : "file";
        if ((fp = fopen(file, "rb")) == NULL) {
                perror(file);
                exit(1);
        }
        (void)fseek(fp, 0L, SEEK_END);
        printf("tell = %ld¥n", ftell(fp));
        rewind(fp);
        printf("tell = %ld¥n", ftell(fp));
        (void)fclose(fp);
        exit(0);
}
```

rgb

レベル3

書　式

#include <graph. h>

int rgb(r, g, b) ;
int r ; /*　赤　*/
int g ; /*　緑　*/
int b ; /*　青　*/

機　能

赤r、緑g、青bの各成分に応じた色コードを得ます。
なお、色コードは0〜65534の偶数となります。
r、g、bはそれぞれ、0〜31を指定します。

戻り値

色コードを返します。

プログラム例

```
#include        <basic0.h>
#include        <graph.h>


main()
{
        void    test(int);
        int     r, g, b;
        int     d;

        b_init();
        screen(1, 3, 1, 1);
        vpage(1);
        for (r = 0; r < 16; r++) {
                for (g = 0; g < 16; g++) {
                        for (b = 0; b < 16; b++) {
                                for (d = 0; d < 2; d++)
                                        test(rgb(r, g, b) + d);
                        }
                }
        }
        b_exit(0);
}


static void     test(int f)
{
        int     e;

        circle(255, 255, 100, f, 0, 360, 256);
        paint(255, 255, f);
        for (e = 0; e < 200; e++)
                ;
}
```

rmdir

レベル2

書式

#include <direct. h>

int rmdir(pathname) ;
char ＊pathname ; /＊ 削除するディレクトリ名へのポインタ ＊/

機能

pathnameで指定したディレクトリを削除します。
ただし、空でないディレクトリ、カレントディレクトリおよびルートディレクトリを削除することはできません。

戻り値

正常終了ならば正の数（0も）を返します。
エラーならば−1を返します。
また、エラーが発生した場合、errnoと_doserrnoにそれぞれCのエラー番号とOSのエラー番号をセットします。

参照関数

chdir, mkdir

プログラム例

```
#include        <stdio.h>
#include        <direct.h>

#define DIR1    "¥¥tmp2"
#define DIR2    "¥¥tmp2¥¥sub"


main()
{
        if (mkdir(DIR1) == -1)
                printf("create error `%s'.¥n", DIR1);
        else {
                printf("%s is created.¥n", DIR1);
                if (mkdir(DIR2) == -1)
                        printf("create error `%s'.¥n", DIR2);
                else {
                        printf("%s is created.¥n", DIR2);
                        (void)rmdir(DIR2);
                        printf("%s is removed.¥n", DIR2);
                }
                (void)rmdir(DIR1);
                printf("%s is removed.¥n", DIR1);
        }
}
```

rmtemp, rmtmp　レベル2

書　式

```
#include <stdio.h>

int rmtemp( ) ;
int rmtmp( ) ;
```

機　能

tempfile 関数はまた tmpfile 関数によって作成されたテンポラリファイルを削除します。

戻り値

削除したファイルの数を返します。
1つも削除できなかった場合は−1を返します。

参照関数

tempfile, tmpfile, tempnam, tmpnam

プログラム例

```
#include        <stdio.h>


main()
{
        FILE    *fp;

        if ((fp = tempfile("B:", "sample.tmp")) == NULL)
                fprintf(stderr, "can't open temp file¥n");
        else
                rmtemp();
        exit(0);
}
```

rnd

レベル3

書　式	# include <basic. h> double rnd( ) ;
機　能	浮動小数点数値の擬似乱数を求めます。
戻り値	0 以上 1 未満の浮動小数点数の擬似乱数が返ります。

rstmem

レベル2

書　　式

```
#include <stdlib.h>

void rstmem( ) ;
```

機　　能　ヒープ領域のブレーク値（領域の上限値）を初期状態に戻します。
ヒープ領域とは、malloc関数などが返すメモリ領域の全体をいいます。

戻り値　戻り値はありません。

注：malloc関数などによって割りつけられたメモリ領域を解放せずに、本関数を実行してはいけません。

sbrk

レベル 2

書式

#include <stdlib. h>

char　＊sbrk(incr)；
size_t incr；　/＊　ブレーク値に加算するバイト数　＊/

機能

ヒープ領域のブレーク値（領域の上限値）に incr を加算し、ヒープ領域を拡張します。
incr に負の数を指定すると、その後の動作は保障されません。

戻り値

変更前のブレーク値を返します。
エラーが発生した場合は－1 を返し、errno に ENOMEM をセットします。

参照関数

calloc, free, malloc, realloc

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        if (sbrk(100) == (char *)-1)
                printf("Not Success¥n");
        else
                printf("Success¥n");
}
```

scanf

レベル２

書　式

```
#include <stdio.h>

int scanf(format_string 〔, argument…〕);
char   *format_string;  /*  書式文字列へのポインタ  */
```

機　能

標準入力からデータを読み込み、format_string の書式に従って変換し、後に続く argument のフィールドに代入します。
また、引数 argument は format_string の各変換指定に対応する型変数のポインタです。
format_string は次の文字のうち、１つ以上を含まなくてはなりません。

●空白文字(空白'　'、タブ'¥t'、改行'¥n'、垂直タブ'¥v'、改頁'¥f'、復帰'¥r')
　空白文字を読み飛ばします。

●通常文字
　一致した文字のみ、読み込みます。
　標準入力から読み込まれた次の文字が一致しなかった場合には、scanf は終了します。

●パーセント（%）
　次の書式で変換を指定します。
　%〔*〕〔length〕type

【*】

入力を読み飛ばし、データを代入しません。
また、引数を必要としません。

【length】

読み込むデータの最大長を 10 進数の整数で指定します。
最大長を超えたデータは切り捨てられます。
最大長より短いデータや指定したフォーマットでは変換できない文字を読み込んだ場合、空白文字を読み込んだ場合は、length よりも短い文字数を読み込みます。

【type】

変換書式を指定します。

typeの値	入力の型	引数の型
d ld	10進の整数	intまたはlongを指すポインタ
hd	10進の整数	shortを指すポインタ
i li	整数 2、8、16、10進すべて可	intまたはlongを指すポインタ
hi	整数 2、8、16、10進すべて可	shortを指すポインタ
b lb	2進の整数	intまたはlongを指すポインタ
hd	2進の整数	shortを指すポインタ
o lo	8進の整数	intまたはlongを指すポインタ
ho	8進の整数	shortを指すポインタ
x lx	16進の整数	intまたはlongを指すポインタ
hx	16進の整数	shortを指すポインタ
u lu	符号なしの10進整数	unsigned intまたはunsigned longを指すポインタ
hu	符号なしの10進整数	unsigned shortを指すポインタ
e f g	符号つきの浮動小数点値	floatを指すポインタ
le lf lg	符号つきの浮動小数点値	doubleを指すポインタ

typeの値	入力の型	引数の型
c	文字 通常は読み飛ばされる空白文字が、この“c”の指定によって読み込むことが可能 次の空白文字でない文字を読み込むには“% 1S”を使う	入力文字列を格納できる大きさをもつ文字配列を指すポインタ
s	文字列	入力文字列とヌル文字‘¥0’を格納できる大きさをもつ文字配列を指すポインタ
p	アドレスを表す16進数	ポインタを指すポインタ
n	読み込みは行なわない 現在までに入力から読み込んだ文字数を、引数が指すint型の変数に代入する	intを指すポインタ

空白文字以外によって区切られている文字列を読み込むときは、大カッコ〔 〕の中の文字セットをs（文字列）型文字の代わりに使います。
対応する入力フィールドは大カッコにはない文字を発見するまで読み出します。
また、最初の文字がキャレット（^）の場合は、反対の操作を行います。
入力フィールドは残りの文字セットの最初の文字を発見するまで読み出します。
ヌル文字‘¥0’を除いて文字列を格納するときは、”%nc”（nは10進数の整数）とします。
%cに対応する引数は文字配列を指すポインタで、n個の文字を標準入力から読み出し、ヌル文字‘¥0’はつけ加えません。
%pに対応する引数はポインタを指すポインタで、printfの%p変換によって出力されたデータを読む込みための書式です。
%nに対応する引数は、intを指すポインタで、現在までに入力から読み込んだ文字数が、引数のポインタへ代入されます。
このとき、入力は行なわれません。
scanfは次の場合、空白文字を読み出す前に読み込みを終了します。

- lengthで指定した文字数に達したとき
- 指定どおりに変換できない文字を読み出したとき
- 制御文字列と矛盾する文字を読み出したとき

●指定した文字セットにない文字を読み出したとき

戻り値 代入したフィールドの数を返します。

%＊によって代入しなかったフィールドは含みません。

また、%n による代入も含みません。

フィールドに1つも代入しなかった場合は0を返します。

読み込みの前に入力が尽きてしまった場合は EOF を返します。

参照関数 fscanf, printf, sscanf

プログラム例

```
#include        <stdio.h>


main()
{
        char    str[256];
        char    c;
        long    l;
        float   d;

        scanf("%s¥n", str);
        scanf("%c¥n", &c);
        scanf("%ld¥n", &l);
        scanf("%g", &d);
        printf("%s %c %ld %g¥n", str, c, l, d);
}
```

screen

レベル３

書　式

```
#include <basic0.h>

void screen(size, mode, res, disp_sw) ;
int size; /*  表示画面サイズ  */
int mode; /*  実画面・色モード（ページ） */
int res; /*  解像度  */
int disp_sw; /*  グラフィック画面使用可・不可  */
```

機　能

テキストおよびグラフィック画面の初期化を行います。
size には表示画面サイズを指定します。

0：グラフィック 256×256、テキスト 32×16
1：グラフィック 512×512、テキスト 64×32
2：グラフィック 768×512、テキスト 96×32

mode には実画面・色モード（ページ）を指定します。

mode	実画面	色モード（ページ）
0	1024×1024	16色（0）
1	512×512	16色（0～3）
2	512×512	256色（0～1）
3	512×512	65536色（0）

size に2を指定した場合は、mode は0のみが指定可能です。
res には解像度を指定します。

0：低解像度
1：高解像度

size に2を指定した場合は、res は1のみが指定可能です。
disp_sw にはグラフィック画面の使用可・不可を指定します。

0：グラフィック画面使用不可
1：グラフィック画面初期化

disp_sw に 0 を指定するとグラフィックを非表示に、1 を指定すると表示状態になります。

screen 関数を実行すると、グラフィックのクリッピング領域は表示画面サイズと同一となり、テキスト画面のスクロール行は、画面全体となりホームポジションおよびカーソル位置は（0, 0）に設定されます。

また、テキスト画面は常に表示状態となり、スプライト画面は sp_disp（1）を実行するまで表示されません。

戻り値 戻り値はありません。

setbuf

レベル2

書　式

```
#include <stdio.h>

void setbuf (stream, buffer) ;
FILE   *stream ;  /*  FILE構造体へのポインタ  */
char   *buffer ;  /*  ユーザー定義バッファへのポインタ  */
```

機　能

streamで指定するストリームが使用する入出力バッファを、システムが自動的に割りあてたものから任意にユーザーが定義したものに変更します。
bufferは新しい入出力バッファを指すポインタで、バッファのサイズは定数BUFSIZでなければいけません。
NULLを指定するとstreamはバッファリングされません。
通常stderrやstdauxなどはバッファリングしませんが、setbufによりバッファリングすることができます。

戻り値

戻り値はありません。

参照関数

fflush, fopen, fclose

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

static char     buf[BUFSIZ];


main(int argc, char *argv[])
{
        char    *file;
        FILE    *fp;
        int     c;

        file = (argc > 1) ? argv[1] : "file";

        if ((fp = fopen(file, "r")) == NULL) {
                perror(file);
                exit(1);
        }
        fprintf(stderr, "buffering on¥n");
        setbuf(fp, buf);
        while ((c = getc(fp)) != EOF)
                putchar(c);
        (void)fclose(fp);

        if ((fp = fopen(file, "r")) == NULL) {
                perror(file);
                exit(1);
        }
        fprintf(stderr, "buffering off¥n");
        setbuf(fp, NULL);
        while ((c = getc(fp)) != EOF)
                putchar(c);
        (void)fclose(fp);

        exit(0);
}
```

setjmp

レベル2

書　式

```
#include <setjmp.h>

int setjmp(env);
jmp_buf env;  /* 現在のスタック環境をセーブする領域 */
```

機　能

現在のスタックを env にセーブします。
env は longjmp 関数に引きわたされます。
setjmp および longjmp 関数は非ローカル goto を実行し、エラーの操作などにおいて、通常の関数呼び出し、および関数からの復帰以外のシーケンスを実行します。
setjmp が呼び出されると、現在のスタック環境が env にセーブされ、逆に longjmp 関数が呼び出されると、セーブした環境のリストアが行われ、対応する setjmp 呼び出しの直後に制御が戻ります。
longjmp 関数は、setjmp を呼び出す前に呼び出してはいけません。
また、setjmp を呼び出した関数が return 文を実行した後や、関数から抜け出したりした後で longjmp 関数をしてはいけません。
ほかに、setjmp 呼び出し時と longjmp 関数呼び出し時でスーパーバイザ/ユーザーモードが異なってはいけません。
もし、上記の禁止事項が守られていない状態で longjmp 関数を行うと、その後のプログラムの動作は保障されません。

戻り値

通常の呼び出しによってスタック環境がセーブされた場合は0を返します。
longjmp 関数の呼び出しによってスタック環境がリストアされ、setjmp に制御が移行してきた場合は0以外（longjmp 関数から受け渡された値）を返します。

参照関数

longjmp

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <setjmp.h>

static jmp_buf  label;


main()
{
        void    func1(void);
        void    func2(void);
        int     r;

        if ((r = setjmp(label)) == 0)
                puts("setjmpを実行しました。");
        else if (r == 1)
                puts("longjmpによってfunc1()から来ました。");
        else if (r == 2)
                puts("longjmpによってfunc2()から来ました。");
        func1();
        func2();
        exit(0);
}


static void     func1(void)
{
        static int      n = 0;

        if (n++ < 3)
                longjmp(label, 1);
}


static void     func2(void)
{
        static int      n = 0;

        if (n++ < 2)
                longjmp(label, 2);
}
```

setmode

レベル2

書　式

```
#include <fcntl.h>
#include <io.h>

int setmode(file, mode);
int file;  /*  該当ファイルハンドル  */
int mode;  /*  ファイルの変換モード  */
```

機　能

fileで指定されたファイルのファイル変換モードを変更します。
変更できるモードは次の通りです。

modeの値	内　容
O_TEXT	テキストモードに変更する 入力時には復帰改行'¥r/¥n'を改行'¥n'に変換し、出力時には改行'¥n'を復帰改行'¥r/¥n'に変換する
O_BINARY	バイナリモードに変更する 上記の変換は行われない

通常setmodeはstdin、stdout、stdaux、stdprnのデフォルトのモードを変更するために使用しますが、他のファイルに対しても使用することができます。

戻り値

正常に変更した場合は、前の変換モードを返します。
エラーが発生した場合には−1を返しerrnoに下記のどちらかをセットします。

errnoの値	内　容
EBADF	指定したファイルが無効
EINVAL	modeの指定の誤り

参照関数

creat, fopen, open

プログラム例

```
#include        <stdio.h>
#include        <fcntl.h>
#include        <io.h>

#define SIZE    512


main()
{
        int     ifn, ofn;
        char    buf[SIZE];
        int     n;
        int     c;

        ifn = fileno(stdin);
        ofn = fileno(stdout);
        setmode(ifn, O_BINARY);
        setmode(ofn, O_BINARY);
        while ((n = read(ifn, buf, SIZE)) > 0)
                write(ofn, buf, n);
}
```

setmspos

レベル3

書　式

#include <mouse. h>

int setmspos(x, y) ;
int x ; /*　マウスカーソルのX座標　*/
int y ; /*　マウスカーソルのY座標　*/

機　能

指定した座標にマウスカーソルを移動します。
xにはマウスカーソルのX座標をを指定します。
yにはマウスカーソルのY座標を指定します。
x、yに指定できる範囲は、msarea関数によって設定された範囲内です。

戻り値

終了コードを返します。
0ならば正常終了、0以外ならばエラーです。

プログラム例

```
#include        <stdio.h>
#include        <conio.h>
#include        <basic0.h>
#include        <mouse.h>
#include        <graph.h>


main()
{
        int     x, y;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);
        mouse(0);
        msarea(0, 0, 767, 511);
        box(0, 0, 767, 511, 15, 0xffff);
        mouse(1);
        for (;;) {
                cputs("mouse cursor area 'x' (0-767)¥r¥n");
                cputs("mouse cursor area 'y' (0-511)¥r¥n");
                cputs("mouse cursor position (x,y) ?");
                scanf("%d,%d", &x, &y);
                setmspos(x, y);
        }
        b_exit(0);
}
```

setvbuf, setnbf

レベル 2

書式

```
#include <stdio.h>

int setvbuf(fp, buff, buftype, bufsize) ;
void setnbf(fp) ;

FILE  *fp;  /*  FILE構造体へのポインタ  */
char  *buff;  /*  バッファへのポインタ  */
int buftype;  /*  バッファリングの方法  */
size_t bufsize;  /*  バッファサイズ  */
```

機能

setvbufは、fpで指定するストリームのバッファを、最初に確保したものからbuffに変更します。
bufsizeには、使用するバッファの大きさをバイト数で指定します。
buftypeには、バッファリングの方法によって、次の定数を指定します。

buftypeの値	内容
_IOFBF	入出力ともバッファリング
_IOLBF	出力を行単位でバッファリング
_IONBF	バッファリングなし

buffにNULLを指定すると、自動的にsizeの大きさのメモリを確保してそれを使用します。
setnbfは、fpで指定するストリームのバッファリング処理を1文字単位で実行するように変更します。
これらの関数は、必ずファイルをオープンした直後に実行しなければなりません。

戻り値

setvbufは正常ならば0を、エラーならば0以外を返します。
setnbfは、戻り値はありません。

sgn

レベル 3

書式

```
#include <basic.h>

double sgn(val);
double val;
```

機能

数値 val の符号を調べた結果を double 型で得ます。

戻り値

符号によって、次の値を返します。

符号	戻り値
＋	1
0	0
－	－1

signal

レベル 2

書　　式

```
#include <signal.h>

void  (* signal(sig, func))(int);
int sig;  /*  シグナル値  */
void  (* func)(int);  /*  シグナルを受け取った時の処理を行う関数へのポ
                          インタ  */
```

機　　能

割り込みシグナルを受け取った時の処理を設定します。
sig は signal. h で定義されたマクロ SIGINT, SIGABRT, SIGTERM のいずれかを指定します。
sig の意味を以下に示します。

sig の値	意　　味
SIGINT	CTRL+C、BREAK キーによる割り込み
SIGABRT	異常終了
SIGTERM	プログラム終了の要求

SIGABRT, SIGTERM は、raise 関数によってシグナルを送らない限り発生しません。
また、CTRL+C、BREAK キーによる SIGINT シグナルでは、func を呼び出したのち、必ずそのシグナルのデフォルトの処理が行われます。
raise 関数によって送られた SIGINT シグナルでは、この signal で設定された func が行う処理のみが行われます。
func は signal. h で定義されたマクロ SIG_IGN, SIG_DFL、または関数へのポインタを指定します。
func 関数の引き数は、int が 1 つで、通常、戻り値はあってはいけません。
func の意味を次頁に示します。

funcの値	内　　容
SIG_IGN	割り込みシグナルを無視する
SIG_DFL	呼び出し中のプロセスを終了し、制御をOSに戻す 呼び出しプロセスによってオープンされたファイルは全てクローズするが、バッファはフラッシュしない
関数へのポインタ	funcによって指定した関数を実行する前に、まずシグナルに対する処理をSIG_DFLにセットしてから関数を実行し、その後は割り込みシグナルを受け取った直後のポインタに制御を戻す 関数実行中にsignalによって再設定しない限り割り込みシグナルは、SIG_DELとして扱われる したがって、呼ばれた関数内においてシグナルをリセットすることが可能

戻り値　設定変更前のシグナルを受け取った時の処理の値を返します。
エラーが発生した時はSIG_ERRを返し、sigの値が無効であることを示すEINVALがerrnoにセットされます。

参照関数　abort, exit, _exit, raise, spawnle, spawnlp, spawnnv, spawnve, spawnvp

プログラム例

```
#include        <stdio.h>
#include        <conio.h>
#include        <signal.h>
#include        <stdlib.h>
#include        <process.h>


main()
{
        void    brk(void);

        if (signal(SIGINT, brk) == SIG_ERR) {
                perror("couldn't set SIGINT");
                abort();
        }
        for (;;)
                puts("処理を中断してみてください。");
        exit(0);
}


static void     brk(void)
{
        (void)signal(SIGINT, SIG_IGN);
        flushall();
        cputs("処理を中断されました。");
        cputs("どれかキーを押してください。");
        (void)getch();
        cputs("¥033*");
        exit(1);
}
```

sin, sinh

レベル2

書式

#include <math. h>

double sin(x) ;
double sinh(x) ;
double x ;

機能

sin は x のサインを、sinh は x のハイパボリック（双曲線）サインを計算します。

戻り値

sin は x のサインを返します。
sinh は x のハイパボリックサインを返します。
引数が不正な値の場合は、errno に EDOM がセットされ返り値は不定になります。
オーバーフローが発生した場合は errno に ERANGE がセットされ、返り値は符号が付いた HUGE_VAL になります。
またエラー処理は、matherr 関数を使って行うことができます。

参照関数

acos, asin, atan, atanh, cos, cosh, tan, tanh

プログラム例

```
#include        <stdio.h>
#include        <math.h>

main()
{
        double  x;
        int     i;

        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("sin(%+4.1f) = %g¥n", x, sin(x));
        }
        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("sinh(%+4.1f) = %g¥n", x, sinh(x));
        }
}
```

sizmem

レベル 2

書　式　#include <stdlib.h>

long sizmem( );

機　能　使用可能なメモリブロックに関する情報を通知します。
使用可能なメモリブロックの残りのサイズを、ワード数で通知します。

戻り値　使用可能なメモリブロックの残りのサイズを返します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

main()
{
        long    size;

        size = sizmem();
        printf("使用可能メモリブロックは %ldワード¥n",size);
}
```

spawnl, spawnle, spawnlp, spawnv, spawnve, spawnvp　レベル2

書　式

```
#include <process. h>

int spawnl(mflag, path, arg0, arg1, arg2, …, argn, NULL) ;
int spawnle(mflag, path, arg0, arg1, arg2, …, argn, NULL, envp) ;
int spawnlp(mflag, path, arg0, arg1, arg2, …, argn, NULL) ;
int spawnv(mflag, path, argv) ;
int spawnve(mflag, path, argv, envp) ;
int spawnvp(mflag, path, argv)
int mflag;
char   *path;
char   *arg0, *arg1, *arg2, …, *argn;
char   *argv [  ], *envp [  ];
```

機　能

子プロセスを新たに作成して実行します。
このとき、子プロセスを実行するための十分なメモリが必要です。
mflagの値によって、実行中の動作を決定します。
mflagの値と内容は次の通りです。

mflagの値	内　　容
P_WAIT	子プロセスが終了するまで、親プロセスの動作を保留する

pathには子プロセスとして実行するファイル名を指定します。
拡張子を省略すると、'X'をつけます。
spawnlpとspawnvpは、環境変数pathで設定したディレクトリからファイルを探します。
子プロセスへの引数は、文字列を指すポインタかポインタの配列としてわたします。
パス名と引数をあわせて255バイトまでで指定します。
文字列末のヌル文字もこのバイト数に含まれますので注意してください。
子プロセスへの引数の数があらかじめわかっているときは、spawnl, spawnle, spawnlpを使います。
引数の数がわからない、つまり変数のときは、spawnv, spawnve, spawnvpを使います。
spawnl, spawnlp, spawnv, spawnvpは、親プロセスの環境を引き継ぎます。
spawnle, spawnveでは、envpを使って子プロセスの環境設定を変えることがで

きます。

envpは文字列を指すポインタの配列で、各文字列は、

VALNAME = value

とします。

ここでVALNAMEは環境変数名で、valueはその値です。

envpがNULLのときは、親プロセスの環境を引き継ぎます。

戻り値 正常終了した場合は正の数（0も）を返します。

これは起動したプロセスの終了コードです。

エラーが発生した場合は−1を返します。

その場合errnoに下記の値がセットされます。

erronの値	内容
E2BIG	引数リストが256バイトを超えている
EMFILE	オープンファイルの数が多すぎる
ENOENT	ファイルまたはパス名が見つからない
ENOEXEC	指定のファイルが実行できない
ENOMEM	子プロセスを実行するためのメモリが不足している

参照関数 execl, execle, execlp, execlpe, execv, execve, execvp, execvpe, abort, exit

sp_clr

レベル3

書式

```
#include <sprite.h>

void sp_clr(cd1, cd2) ;
int cd1, cd2 ;  /*  スプライトプレーン番号  */
```

機能

スプライトをクリアします。

cd1、cd2でクリアするスプライトプレーンの範囲を指定します（いずれも0～255）。

cd1に'NASI'を指定すると、cd1は0になります。

cd2に'NASI'を指定すると、cd1のみの指定になります。

cd1、cd2ともに、'NASI'を指定するとcd1は0、cd2は255になります。

戻り値

戻り値はありません。

sp_color

レベル3

書　式

```
#include <sprite.h>

int sp_color(p, c, pb) ;
char p;  /*  パレットコード  */
int c;  /*  カラーコード  */
int pb;  /*  パレットブロック  */
```

機　能

スプライトのパレットの色の設定および読み出しを行います。
pにはパレットコードを指定します（0～15）。
cにはカラーコードを指定します（0～65535）。
cに'NASI'を指定すると、カラーコードの読み出しのみを行います。
pbにはパレットブロックを指定します（1～15）。
pbに'NASI'を指定すると、1を指定したことになります。

戻り値

設定前のカラーコードを返します。

sp_def

レベル3

書　式

```
#include <sprite.h>

void sp_def(cd, ca, psize) ;
char cd; /*  パターンコード  */
char *ca; /*  パターンデータを格納してある領域へのポインタ  */
int psize; /*  パターンデータのサイズ  */
```

機　能

スプライトにパターンを定義します。
cd にはパターンコードを指定します（0～255）。
ca には、パターンデータを格納してある領域へのポインタを指定します。
この領域は、パターンサイズが 8×8 のときは 64 バイト、パターンサイズが 16×16 のときは 256 バイト必要です。
格納しておくデータの 1 バイトが、スプライトパターンの 1 ドットに対応します。
0～15 のデータを格納しておくと、それがスプライトのドットの色になります。
psize にはパターンデータのサイズを指定します（0、1）。
0 を指定するとパターンサイズが 8×8 に、1 を指定するとパターンサイズが 16×16 になります。
'NASI' を指定すると、1 を指定したことになります。

戻り値

戻り値はありません。

sp_disp

レベル3

書　　式

```
#include <sprite.h>

void sp_disp(ch);
char ch;  /*  表示モード  */
```

機　　能

スプライト面（スプライト、バックグラウンド）の表示制御を行います。
chには次の内容を指定します。

0：スプライト、バックグラウンドを表示しない
1：スプライト、バックグラウンドを表示する

戻 り 値

戻り値はありません。

sp_init

レベル３

書　　式　# include <sprite. h>

```
int sp_init( ) ;
```

機　　能　スプライト画面を初期化します。
初期化するものは次の通りです。

スクロールレジスタ
スプライトパターン
スプライト用パレット

戻 り 値　正常に初期化できたときは、0を返します。
エラーのとき（画面サイズが768×512のときなど）は−1を返します。

sp_move

レベル 3

書　　式

```
#include <sprite.h>

void sp_move(s, x, y, cd) ;
char s; /*  プレーン番号  */
int x; /*  X座標  */
int y; /*  Y座標  */
int cd; /*  パターンコード  */
```

機　　能

スプライトパターンをプレーンに登録し、表示位置を決めます。
sにはプレーン番号を指定します（0～127）。
xにはX座標を指定します（－16～1007）。
yにはY座標を指定します（－16～1007）。
cdにはパターンコードを指定します（0～255）。
パターンコードを指定すると、そのスプライトは自動的に表示されます。
x、y、cdに‘NASI’を指定すると、以前に設定されていた値を使用します。

戻 り 値

戻り値はありません。

プログラム例

```
#include        <math.h>
#include        <basic0.h>
#include        <sprite.h>

static char     pat2[256];
static char     pat1[256] =  {
        0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
        0, 0, 2, 2, 2, 3, 3, 3, 3, 3, 3, 2, 2, 2, 0, 0,
        0, 2, 4, 4, 4, 5, 5, 5, 5, 5, 5, 4, 4, 4, 2, 0,
        1, 2, 4, 6, 6, 6, 7, 7, 7, 7, 6, 6, 6, 4, 2, 1,
        1, 2, 4, 6, 8, 8, 9, 9, 9, 9, 8, 8, 6, 4, 2, 1,
        1, 3, 5, 6, 8,10,11,11,11,11,10, 8, 6, 5, 2, 1,
        1, 3, 5, 7, 9,11,12,13,13,12,11, 9, 7, 5, 3, 1,
        1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
        1, 3, 5, 7, 9,11,13,15,15,13,11, 9, 7, 5, 3, 1,
        1, 3, 5, 7, 9,11,12,13,13,12,11, 9, 7, 5, 3, 1,
        1, 3, 5, 6, 8,10,11,11,11,11,10, 8, 6, 5, 2, 1,
        1, 2, 4, 6, 8, 8, 9, 9, 9, 9, 8, 8, 6, 4, 2, 1,
        1, 2, 4, 6, 6, 6, 7, 7, 7, 7, 6, 6, 6, 4, 2, 1,
        0, 2, 4, 4, 4, 5, 5, 5, 5, 5, 5, 4, 4, 4, 2, 0,
        0, 0, 2, 2, 2, 3, 3, 3, 3, 3, 3, 2, 2, 2, 0, 0,
        0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
};


main()
{
        int     x;
        int     y;
        int     i;

        b_init();
        screen(1, 3, 1, 1);
        sp_clr(0, 255);
        sp_off(0, 127);
        sp_disp(1);
        sp_def(0, pat1, 1);
        sp_def(1, pat2, 1);
        for (i = 0; i < 256; i++) {
                x = i * 2 - 16;
                y = sin(2.0 * 3.1416 * x / 256) * 200.0;
                sp_move(0, x, 255 - y, 0);
                sp_move(1, x, 255 + y, 1);
        }
        sp_disp(0);
        b_exit(0);
}
```

sp_off

レベル３

書　式

```
#include <sprite.h>

void sp_off(s1, s2) ;
int s1, s2 ;  /*  スプライトプレーン番号  */
```

機　能

スプライトを非表示状態にします。

s1 から s2 までのプレーンのスプライトを非表示状態にします（いずれも 0 ～127）。

s1 に 'NASI' を指定すると、s1 は 0 になります。

s2 に 'NASI' を指定すると、s1 のみの指定になります。

s1，s2 ともに 'NASI' を指定すると、s1 は 0、s2 は 127 になります。

戻り値

戻り値はありません。

sp_on

レベル3

書式

```
#include <sprite.h>

void sp_on(s1, s2) ;
int s1, s2;  /*  スプライトプレーン番号  */
```

機能

スプライトを表示状態にします。
s1からs2までのプレーンのスプライトを表示状態にします(いずれも0～127)。
s1に'NASI'を指定すると、s1は0になります。
s2に'NASI'を指定すると、s1のみの指定になります。
s1、s2ともに'NASI'を指定すると、s1は0、s2は127になります。

戻り値

戻り値はありません。

sp_pat　レベル 3

書　式

```
#include <sprite.h>

void sp_pat(cd, ca, psize) ;
char cd ; /* パターンコード */
char *ca ; /* パターンデータを格納する領域へのポインタ */
int psize ; /* パターンデータのサイズ */
```

機　能

スプライトのパターンデータを指定した領域へ読み込みます。
cd にはパターンコードを指定します (0～255)。
ca にはパターンデータを読み込む領域へのポインタを指定します。
この領域は、パターンサイズが8×8のときは64バイト、パターンサイズが16×16のときは256バイト必要です。
スプライトパターンの1ドットが1バイトのデータになって格納されます。
スプライトのドットの色が0～15のデータになります。
psize にはパターンデータのサイズを指定します (0、1)。
0を指定するとパターンサイズが8×8に、1を指定するとパターンサイズが16×16になります。
'NASI' を指定すると、1を指定したことになります。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <basic0.h>
#include        <graph.h>
#include        <sprite.h>

#define SIZE    (16 * 16)

static char     pat2[SIZE];
static char     pat1[SIZE];


main()
{
        int     x;
        int     y;
        int     i;

        b_init();
        for (i = 0; i < SIZE; i++) {
                pat1[i] = i / 32;
                pat2[i] = i / 32 + 8;
        }
        screen(1, 3, 1, 1);
        sp_clr(0, 255);
        sp_off(0, 127);
        sp_disp(1);
        sp_def(0, pat1, 1);
        sp_def(1, pat2, 1);
        for (i = 0; i < 16; i++)
                sp_color((char)i, rgb(i * 2, i * 2, i * 2), 1);/* ! */
        sp_move(0, 100, 100, 0);
        sp_move(1, 100, 200, 1);
        sp_pat(0, pat2, 1);
        for (y = 0; y < 16; y++) {
                for (x = 0; x < 16; x++)
                        printf("%d  ", pat2[x + y * 16]);
                putchar('¥n');

        }
        sp_disp(0);
        b_exit(0);
}
```

sprintf

レベル 2

書　式　#include <stdio. h>

int sprintf(buffer, format_string [, argument…]) ;
char　*buffer ;　/*　出力データ格納領域へのポインタ　*/
char　*format_string ;　/*　書式文字列へのポインタ　*/

機　能　format_string で指定した書式に従って出力を変換し、buffer が指す領域に格納します。

format_string に続く引数は書式に従って変換され、出力されます。

format_string の書式と機能は printf 関数と同じです。

詳細については、printf 関数を参照してください。

戻り値　出力した文字数を返します。

参照関数　cprintf, fprintf, printf, sscanf

プログラム例

```
#include        <stdio.h>

static char     buf[256];


main()
{
        sprintf(buf, "hello world");
        puts(buf);
        sprintf(buf, "integer[%d]", 200);
        puts(buf);
        sprintf(buf, "integer[%x]", 200);
        puts(buf);
        sprintf(buf, "float[%f]", 3.1415926);
        puts(buf);
        sprintf(buf, "float[%g]", 3.1415926);
        puts(buf);
        sprintf(buf, "character[%c]", 'A');
        puts(buf);
        sprintf(buf, "string[%s]", "string");
        puts(buf);
}
```

sp_set

レベル3

書　式

```
#include <sprite.h>

int sp_set(c, c1, c2, pcode, prty) ;
char c ; /*  スプライトプレーン番号  */
int x ; /*  X座標  */
int y ; /*  Y座標  */
int pcode ; /*  パターンコード  */
int prty ; /*  プライオリティ  */
```

機　能

スプライトのスクロールレジスタの設定を行います。
cにはプレーン番号を指定します（0～127）。
xにはX座標を指定します（0～1023）。
yにはY座標を指定します（0～1023）。
pcodeには、反転の指定、パレットブロック番号指定、パターンコードの指定をします。
pcodeはビットごとに次の内容と値を設定できます。

ビット16～31	：常に0
ビット15	：垂直反転（0…標準、1…反転）
ビット14	：水平反転（0…標準、1…反転）
ビット12～13	：常に0
ビット8～11	：パレットブロック番号（0～15）
ビット0～7	：パターンコード（0～255）

prtyには表示の優先度を指定します（0～3）。

0：スプライト表示なし
1：バックグラウンド0、バックグラウンド1、スプライトの順
2：バックグラウンド0、スプライト、バックグラウンド1の順
3：スプライト、バックグラウンド0、バックグラウンド1の順

c1, c2, pcode, prtyに'NASI'を指定すると、以前に設定されていた値を使用します。

戻り値

正常終了したときは0を返します。
エラーが発生した（画面サイズが768×512のときなど）ときは−1を返します。

参照関数

sp_stat

sp_stat

レベル 3

書　式

```
#include <sprite.h>

int sp_stat(sp, sel_mode) ;
char sp;  /*  スプライトプレーン番号  */
char sel_mode;  /*  モード  */
```

機　能

指定したスプライトのスクロールレジスタの状態を調べます。
sp にはスプライトプレーン番号を指定します。
sel_mode には調べる内容を指定します。

0：X 座標
1：Y 座標
2：パターンコード
3：プライオリティ

X 座標、Y 座標、パターンコード、プライオリティについての詳細は sp_set を参照してください。

戻り値

正常終了のときは sel_mode に従い、調べた値（正の数）を返します。
エラーが発生したときは負の数を返します。

参照関数

sp_set

sqrt

レベル2

書　式

#include <math. h>

double sqrt(x) ;
double x ;　/*　正の浮動小数点値　*/

機　能

xの平方根を計算します。

戻り値

xの平方根を返します。
引数が不正な値の場合は、errnoにはEDOMがセットされ戻り値は不定になります。
またエラー処理は、matherr関数ルーチンを使って行うことができます。

参照関数

exp, log, matherr, pow

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        int     i;

        for (i = 0; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("sqrt(%+4.1f) = %g¥n", x, sqrt(x));
        }
}
```

srand

レベル 2

書　式

#include <stdlib.h>

void srand(seed) ;
unsigned int seed;

機　能

擬似乱数の発生系列を初期化します。
再初期化する場合は、seed を 1 にします。
seed を 1 以外にすると、その値に対応した初期化が行われます。

戻り値

戻り値はありません。

参照関数

rand

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define MAX     128
#define RANGE   100


main(int argc, char *argv[])
{
        int     i;
        int     s;

        s = (argc == 1) ? 1 : atoi(argv[1]);

        srand(s);

        for (i = 0; i < MAX; i++)
                printf("%10d¥n", rand() % RANGE);
}
```

sscanf

レベル2

書　式

#include <stdio. h>

int sscanf(buffer, format_string [, argument…]) ;
char　*buffer ;　/*　読み込まれる文字列へのポインタ　*/
char　*format_string ;　/*　書式文字列へのポインタ　*/

機　能

buffer が指す文字列を読み込み、format_string の書式に従って変換し、後に続く argument のフィールドに代入します。
format_string の内容は scanf と同じです。
また、引数 argument は format_string の各変換指定に対応する型変数のポインタです。

戻り値

代入したフィールドの数を返します。
代入しなかったフィールドは含みません。
代入したフィールドがない場合は 0 を返します。
読み込みの前に入力が尽きてしまった場合は EOF を返します。

参照関数

fscanf, scanf, sprintf

プログラム例

```
#include        <stdio.h>

static char     buf[256];


main()
{
        char    str[256];
        char    c;
        long    l;
        float   d;

        gets(buf);
        sscanf(buf, "%s¥n", str);
        gets(buf);
        sscanf(buf, "%c¥n", &c);
        gets(buf);
        sscanf(buf, "%ld¥n", &l);
        gets(buf);
        sscanf(buf, "%g", &d);

        printf("%s %c %ld %g¥n", str, c, l, d);
}
```

stat

レベル2

書式

```
#include <stat.h>

int stat(pathname, buffer) ;
char  *pathname ;  /*  情報を得たいファイルのパス名  */
struct stat  *buffer ;  /*  stat 構造体を指すポインタ  */
```

機能

pathnameで指定したファイル、またはディレクトリに関する情報をbufferが指す構造体に格納します。
stat構造体はstat.hで以下のように定義されています。

```
struct  stat  {
        dev_t   st_dev ;    /*  ファイルが存在するドライブの番号  */
        ino_t   st_ino ;    /*  常に0  */
        short   st_mode ;   /*  ファイルタイプとファイル属性
                                ディレクトリならばS_IFDIRのビットが
                                セットされ、ファイルならばS_IFREGの
                                ビットがセットされる
                                アクセスモードによってユーザー（リー
                                ド/ライト）のビットがセットされる
                                実行可能ファイル（.X .R .Z .BAT）なら
                                ばS_IEXECのビットがセットされる
                                */
        short   st_nlink ;  /*  常に1  */
        short   st_uid ;    /*  常に0  */
        short   st_gid ;    /*  常に0  */
        dev_t   st_rdev ;   /*  ファイルが存在するドライブの番号  */
        off_t   st_size ;   /*  ファイルの大きさ（バイト数）  */
        time_t  st_atime ;  /*  最終変更日時。time関数の返り値と同じ
                                表現  */
        time_t  st_mtime ;  /*  最終変更日時。time関数の返り値と同じ
                                表現  */
        time_t  st_ctime ;  /*  最終変更日時。time関数の返り値と同じ
                                表現  */
} ;
```

戻り値

ステータス情報を正しく得た場合には0を返します。
エラーが発生した場合は−1を返し、ENOENTをerrnoにセットします。

参照関数　access, fstat

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <stat.h>


main(int argc, char *argv[])
{
        struct stat     inf;
        char    *file;

        file = (argc > 1) ? argv[1] : "file";
        if (stat(file, &inf) == -1) {
                perror(file);
                exit(1);
        }

        printf("name = %s, size = %d¥n", file, inf.st_size);

        exit(0);
}
```

stcgfe, stcgfn

レベル2

書　式

```
#include <string.h>

int stcgfe(ext, name);
char  *ext;  /*  拡張子を格納する領域へのポインタ  */
char  *name; /*  ファイル名へのポインタ  */

int stcgfn (node, name);
char  *node; /*  主ファイル名を格納する領域へのポインタ  */
char  *name; /*  ファイル名へのポインタ  */
```

機　能

ファイル名の解析を行います。
stcgfe は、name が指すファイル名から拡張子をとり出し、ext に格納してその文字数を返します。
stcgfn は、name が指すファイル名から主ファイル名（'.'と拡張子を除いたもの）をとり出し、node に格納してその文字数を返します。

戻り値

拡張子、または主ファイル名の文字数を返します。

stick

レベル３

書　　式

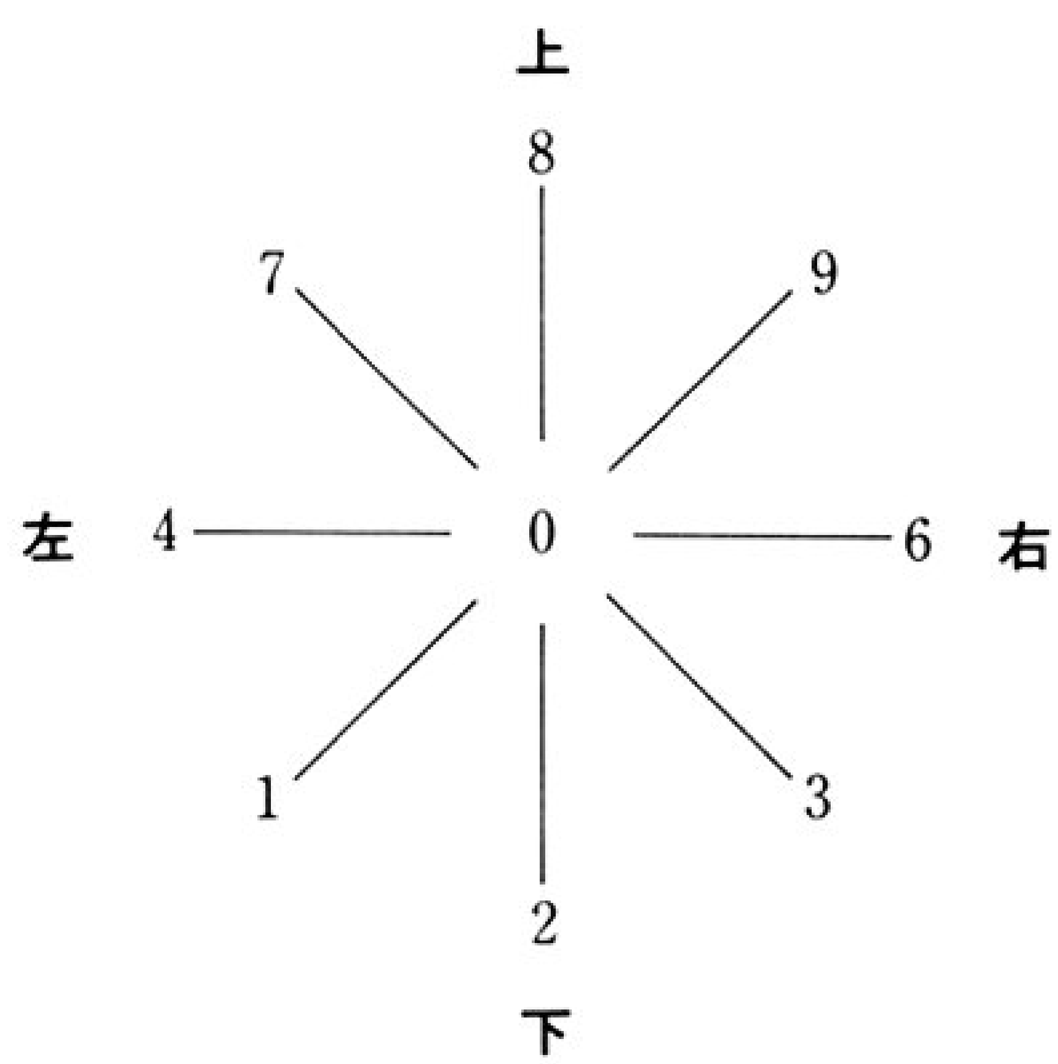
```
#include <stick.h>

int stick(stickno);
int stickno;  /*  ジョイスティック番号  */
```

機　　能　ジョイスティックの状態を調べます。
stickno に 1 を指定するとジョイスティック 1 を、2 を指定するとジョイスティック 2 を調べます。

戻 り 値　状態によって、次の値を返します。

スティック位置	戻り値
中央	0
左後	1
後	2
右後	3
左	4
右	6
左前	7
前	8
右前	9



［ジョイスティックの状態と戻り値］

strbpl

レベル 2

書　式

```
#include <string.h>

int strbpl(s, max, t)
char  *s[ ]; /* ポインタ配列  */
int max; /*  配列の要素数  */
char  *t; /*  文字列を指すポインタ  */
```

機　能

tが指す文字列の並びをもとに、ポインタ配列をsに作成します。

tは、それぞれヌル文字'¥0'で終了する文字列の並びであり、tの最後にはもう1つヌル文字がなければなりません。

maxには、配列sの要素数を指定します。

tに含まれる文字列の数がmaxより大きいか等しい場合、strbplは−1を返します。

配列sの各要素には、tに含まれる文字列へのポインタが代入され、文字列が尽きると終りを表すNULLを要素に代入します。

したがって配列の要素数は文字列の数より1つ多くなければなりません。

戻り値

ポインタの項目数を返します。

項目数がmaxより大きいか等しいときには−1を返します。

プログラム例

```
#include        <stdio.h>
#include        <string.h>

#define MAX              16

char    t[] = "Sharp¥0Computer¥0x68000¥0C compiler¥0PRO 68K¥0";


main()
{
        char    *s[MAX];
        int     i;
        int     n;

        n = strbpl(s, MAX, t);

        for (i = 0; i < n; i++)
                printf("%s¥n", s[i]);

}
```

strcat, strchr, strcmp, strcmpi, stricmp, strcpy, strcspn, strdup　レベル2

書　式

```
#include <string.h>

char  *strcat(string1, string2);
char  *string1;  /*  連結される文字列を指すポインタ  */
char  *string2;  /*  連結する文字列を指すポインタ  */

char  *strchr(string, c);
char  *string;  /*  検索が行われる文字列を指すポインタ  */
int c;  /*  検索する文字  */

int strcmp(string1, string2);
char  *string1, *string2;  /*  比較が行われる2つの文字列を指すポインタ  */

int strcmpi(string1, string2);
char  *string1, *string2;  /*  比較が行われる2つの文字列を指すポインタ  */

int stricmp(string1, string2);
char  *string1, *string2;  /*  比較が行われる2つの文字列を指すポインタ  */

char  *strcpy(string1, string2);
char  *string1;  /*  複写先文字列を指すポインタ  */
char  *string2;  /*  複写元文字列を指すポインタ  */

size_t strcspn(string1, string2);
char  *string1;  /*  検索が行われる文字列を指すポインタ  */
char  *string2;  /*  文字セットを指すポインタ  */

char  *strdup(string);
char  *string;  /*  複写を行う文字列を指すポインタ  */
```

機　　能	末尾がヌル文字'¥0'の文字列に対して次の操作を行います。

【strcat】

string1のうしろにstring2を連結して1つの文字列にして、末尾にヌル文字'¥0'を付加します。

【strchr】

string内で文字cを検索します。
なお、ヌル文字'¥0'も検索することができます。

【strcmp】

string1とstring2を文字コードの順序で比較します。
大文字と小文字は区別されます。

【strcmpi】

string1とstring2を文字コードの順序で比較します。
大文字と小文字は区別されません。

【stricmp】

string1とstring2を文字コードの順序で比較します。
大文字と小文字は区別されません。

【strcpy】

文字列string2（末尾のヌル文字を含む）を文字列を格納する領域string1にコピーします。

【strcspn】

string1先頭から、string2に含まれない文字が何文字続いているかを調べます。

【strdup】

領域を自動的に割りつけて、その領域にstringをコピーします。

戻 り 値	以下の戻り値を返します。

【strcat】

連結された文字列を指すポインタを返します。

【strchr】

文字 c の位置を指すポインタを返します。

文字がみつからなかった場合は NULL を返します。

【strcmp】

戻り値とその内容は次の通りです。

戻り値	内　　　容
負	string1 は string2 より小さい
0	string1 と string2 は等しい
正	string1 は string2 より大きい

【strcmpi】

strcmp と同じです。

【stricmp】

strcmp と同じです。

【strcpy】

string1 を指すポインタを返します。

【strcspn】

何文字続いていたかをバイト数で返します。

最初から string2 に含まれる文字があれば 0 を返します。

すべてが string2 に含まれていない文字ならば、string1 の全長（ヌル文字は含まない）を返します。

【strdup】

割りつけられた領域を指すポインタを返します。

割りつけができなかった場合には NULL を返します。

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     p[256];


main()
{
        printf("p                          -> %s¥n",
         strcpy(p, "Sharp"));
        printf("p                          -> %s¥n",
         strcat(p, "x68000"));
        printf("strchr(p, '6')             -> %s¥n",
         strchr(p, '6'));
        printf("strcmp(p, ¥"sharp¥")         -> %d¥n",
         strcmp(p, "sharp"));
        printf("strcmpi(p, ¥"sharp¥")        -> %d¥n",
         strcmpi(p, "sharp"));
        printf("strcspn(p, ¥"0123456789¥") -> %d¥n",
         strcspn(p, "0123456789"));
}
```

strerror

レベル2

書　式

#include <string.h>

char　*strerror(errnum)；
int errnum；　/*　エラー番号　*/

機　能

エラー番号 errnum に対応するエラーメッセージへのポインタを返します。
errnum は、error.h において定義されている番号のみが指定できます。
strerror はマクロで定義されていますので、返される文字列を書きかえてはいけません。
返された文字列を加工したい場合は、いったん他のバッファへコピーしてから、コピーしたものを加工してください。

戻り値

エラーメッセージへのポインタを返します。
定義されていないエラー番号を指定すると、戻り値は不定です。

プログラム例

```
#include        <stdio.h>
#include        <string.h>
#include        <error.h>


main()
{
        void    sub(FILE *, FILE *);

        sub(stdin, stdout);
}


static void     sub(FILE *fp, FILE *ofp)
{
        int     c;

        while ((c = getc(fp)) != EOF)
                putc(c, ofp);
        if (ferror(ofp)) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                clearerr(ofp);
        }
}
```

strftime

レベル2

書　式

```
#include <time.h>

size_t strftime(buf, max, format, time_ptr) ;
char   *buf ;  /*  文字列領域へのポインタ  */
size_t max ;  /*  作成する文字列領域のバイト数  */
char   *format ;  /*  書式制御文字列へのポインタ  */
struct tm   *time_ptr;  /*  変換される時間情報へのポインタ  */
```

機　能

time_ptr が指す構造体に分解された時間情報を、format が指す書式にしたがって文字列に変換し、buf が指す領域に作成します。

max は作成する文字列領域のバイト数を指定します。

max は終端のヌル文字'¥0'を含めたバイト数を指定するので、返される文字列の最大長は max－1 になります。

format には、以下に示す書式制御文字があります。

書式	意　味
%A	曜日の名前
%a	省略された曜日の名前
%B	月の名前
%b	省略された月の名前
%c	citime 関数の返す形式と同じ形式の曜日、日付、時間。
%d	日を表す 01～31 の数字
%H	24 時間制での時間を表す 00～23 の数字
%I	12 時間制での時を表す 00～12 の数字
%j	1月1日からの通算日数を表す 001～366 の数字
%M	分を表す 00～59 の数字
%m	月を表す 01～12 の数字
%p	午前、午後を表す AM または PM の 2 文字
%S	秒数を表す 00～59 の数字

書式	意　味
%U	日曜を週の初めとする1月1日からの通算週を表す00～53の数字
%W	月曜を週の初めとする1月1日からの通算週を表す00～53の数字
%w	日曜を0とする曜日を表わす0～6の数字
%X	時：分：秒で表す時刻
%x	月/日/年で表す日付
%Y	年を表す世紀部分を付け加えた1980～2079の数字
%y	年を表す世紀部分を取り除いた00～99の数字
%Z	地域（時間帯）を表すJSTの文字列
%%	%1文字

戻り値　変換した文字列の長さを返します。
0が返された場合は、変換結果がmaxを超えてしまう場合で、文字列領域の内容は不定になります。

参照関数　asctime, ctime, gmtime, localtime

プログラム例

```
#include        <stdio.h>
#include        <time.h>

#define ARB     128
#define FORM1   "%a %b %d %H:%M:%S %Z %Y"
#define FORM2   "%B %m '%y (%H:%M:%S)"
#define FORM3   "%Y/%m/%d %H:%M:%S"
#define FORM4   "%y年%m月%d日  %p%I時%M分%S秒"


main()
{
        char    buf[ARB];
        time_t  tim;
        struct tm       *tp;

        time(&tim);
        tp = localtime(&tim);

        strftime(buf, ARB, FORM1, tp);
        printf("%s¥n", buf);
        strftime(buf, ARB, FORM2, tp);
        printf("%s¥n", buf);
        strftime(buf, ARB, FORM3, tp);
        printf("%s¥n", buf);
        strftime(buf, ARB, FORM4, tp);
        printf("%s¥n", buf);
}
```

strig

レベル2

書　式

```
#include <stick.h>

int strig(no) ;
int no;  /*  ジョイスティック番号  */
```

機　能

ジョイスティックのトリガボタンの状態を調べます。
no には、次の値を指定します。

1：ジョイスティック1
2：ジョイスティック2

戻り値

エラーのときは−1を、そうでなければ、0から3の値を返します。
それぞれの値の意味は次の通りです。

0：どちらのボタンも押されていない
1：1のボタンだけ押されている
2：2のボタンだけ押されている
3：両方とも押されている

プログラム例

```
#include        <stdio.h>
#include        <basic0.h>
#include        <stick.h>


main()
{
        int     r;

        b_init();
        cls();

        for (;;) {
                locate(10, 10);
                puts("push trigger!");
                r = strig(1);
                locate(10, 11);
                if (r == 0)
                        ;
                else if (r == 1)
                        puts("'A' trigger on!");
                else if (r == 2)
                        puts("'B' trigger on!");
                else if (r == 3) {
                        puts("'A' & 'B' triggers on!");
                        break;
                }
        }
        b_exit(0);
}
```

strins

レベル２

書　　式

#include <string. h>

void strins(source, dest) ;
char　＊source ;　/＊　挿入する文字列へのポインタ　＊/
char　＊dest ;　/＊　挿入先文字列へのポインタ　＊/

機　　能

dest が指す文字列の前に、source が指す文字列を挿入し、1 つの文字列とします。

dest が指す文字列の領域は十分な大きさがなければなりません。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     dst[256] = "x68000";


main()
{
        puts(dst);
        strins("Computer ", dst);
        puts(dst);
        strins("Sharp ", dst);
        puts(dst);
}
```

strlen

レベル2

書　式

```
#include <string.h>

size_t strlen(str);
char   *str;  /*  文字列  */
```

機　能

文字列strの長さをバイト数で求めます。
このとき、文字列の最後のヌル文字'¥0'は長さに含めません。

戻り値

strの長さを返します。

strlwr

レベル 2

書式	#include <string. h> char　*strlwr(str)； char　*str；　/*　文字列　*/
機能	文字列 str のすべての英大文字を英小文字に置き換えます。 他の文字は変換しません。
戻り値	str を指すポインタを返します。
参照関数	strupr

strmfe

レベル2

書　式

include <string. h>

void strmfe(new, old, ext) ;
char　*new ;　/*　新ファイル名格納領域へのポインタ　*/
char　*old ;　/*　旧ファイル名へのポインタ　*/
char　*ext ;　/*　置き換える拡張子へのポインタ　*/

機　能

与えられた要素から、次の形式のファイル名を組み立てます。

drive : ¥path¥node.ext

strmfe は、old が指すファイル名の拡張子を、ext が指す拡張子で置き換え、new が指す領域に格納します。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     new[256];
static char     old[] = "sample.c";
static char     ext[] = "cc";


main()
{
        printf("Old = %s¥n", old);
        strmfe(new, old, ext);
        printf("New = %s¥n", new);
}
```

strmfn, strmfp

レベル２

書式

```
#include <string.h>

void strmfn(fname, drv, path, node, ext) ;
char  *fname, *drv, *path, *node, *ext;

void strmfp(fname, path, node) ;
char  *fname, *path, *node;
```

機能

strmfnは、drv、path、node、extの各引数からファイル名を組み立てて、fnameに格納します。
引数との対応は次の通りです。

drv　：ドライブ名
path ：パス名
node ：主ファイル名
ext　：拡張子

strmfnでは、それぞれの引数がNULLかどうかを調べます。
NULLでなければ、drvの後には":"を、pathの後には"¥"を加えます。
また、nodeがNULLでなければ、nodeの後に"."を加えます。

strmfpは、pathとnodeからファイル名を組み立てて、fnameに格納します。
引数との対応は次の通りです。

path ：パス名（ドライブ名を含む）
node ：主ファイル名

もし、pathがNULLであるか、pathの最後が"¥"、":"でないときは、pathの後に"¥"を加えます。

戻り値

戻り値はありません。

参照関数

strmfe

strncpy, strncat, strncmp, strnset

レベル2

書　　式　#include <string. h>

```
char　*strncat(string1, string2, n) ;
char　*string1 ; /*　文字の追加が行われる文字列へのポインタ　*/
char　*string2 ; /*　追加文字を含む文字列のポインタ　*/
size_t n ; /*　追加する文字数　*/

int strncmp(string1, string2, n) ;
char　*string1, *string2 ; /*　比較が行われる2つの文字列を指すポインタ　*/
size_t n ; /*　比較する文字数　*/

char　*strncpy(string1, string2, n) ;
char　*string1 ; /*　複写先文字列を指すポインタ　*/
char　*string2 ; /*　複写元文字列を指すポインタ　*/
size_t n ; /*　複写する文字数　*/

char　*strnset(sting, c, n) ;
char　*string ; /*　文字の置き換えが行われる文字列へのポインタ　*/
char c ; /*　置き換え文字　*/
size_t n ; /*　置き換えを行う文字数　*/
```

機　　能　末尾がヌル文字'¥0'の文字列に対して、次の操作を行います。

【strncat】

string1 の後ろに string2 の先頭から n 文字を追加し、末尾にヌル文字を付加します。

【strncmp】

string1 と string2 の先頭から n 文字を文字コードの順序で比較します。
大文字と小文字は区別されます。

【strncpy】

文字列 string2 の先頭から n 文字を string1 に複写します。
string2 の長さが n よりも小さい場合は、string1 の長さが n になるまで末尾にヌル文字を付加します。
string2 の長さが n よりも大きい場合は、ヌル文字は付加されません。

【strnset】

stringの最初からn文字分を文字cに置き換えます。

stringの長さがnよりも小さい場合はすべての文字がcに置き換えられます。

戻り値 以下の戻り値を返します。

【strncat】

新しい文字列を指すポインタを返します。

【strncmp】

戻り値とその内容は次の通りです。

戻り値	内容
負	string1はstring2より小さい
0	string1とstring2は等しい
正	string1はstring2より大きい

【strncpy】

string1を指すポインタを返します。

【strnset】

新しい文字列を指すポインタを返します。

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     p[256];


main()
{
        printf("p                            -> %s¥n",
         strncpy(p, "Sharp", 5));
        printf("p                            -> %s¥n",
         strncat(p, "x68000", 6));
        printf("strncmp(p, ¥"sharp¥")          -> %d¥n",
         strncmp(p, "sharp", 5));
        printf("strnset(p, 'x', 5)          -> %s¥n",
         strnset(p, 'x', 5));
}
```

strpbrk

レベル2

書　式　#include <string. h>

```
char  *strpbrk(string1, string2) ;
char  *string1 ;  /*  検索が行われる文字列へのポインタ  */
char  *string2 ;  /*  検索する文字セットへのポインタ  */
```

機　能　文字列 string1 の中で文字列 string2 に含まれている文字を検索し、最初に出現する位置を求めます。
ただし、ヌル文字'¥0'は検索できません。

戻り値　出現位置を指すポインタを返します。
該当する文字がみつからない場合には NULL を返します。

参照関数　strchr, strrchr

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     buf[] = "Sharp x68000";
static char     set[] = "0123456789";


main()
{
        printf("buf = %s¥n", buf);
        printf("set = %s¥n", set);
        printf("result = %s¥n", strpbrk(buf, set));
}
```

strrchr

レベル2

書　式

```
#include <string.h>

char  *strrchr(string, c) ;
char  *string ;  /*  検索が行われる文字列のポインタ  */
int c ;  /*  検索する文字  */
```

機　能

文字列 string の中で文字 c を検索し、最後に出現する位置を求めます。
ヌル文字'¥0'も検索できます。

戻 り 値

出現位置を指すポインタを返します。
発見できなかった場合は NULL を返します。

参照関数

strchr, strpbrk

プログラム例

```
#include        <stdio.h>
#include        <string.h>


main()
{
        char    *str;
        int     c;

        str = "This is a pen.";
        c = 'i';
        printf("str       ->%s¥n", str);
        printf("c         ->%c¥n", c);
        printf("strchr()  ->%s¥n", strchr(str, c));
        printf("strrchr()->%s¥n", strrchr(str, c));
}
```

strrev　　レベル２

書　式

#include <string. h>

char　＊strrev(string)；
char　＊string；　/＊　並び順の置き換えを行う文字列へのポインタ　＊/

機　能

指定した文字列 string の文字列の並び順を逆にします。
ただしヌル文字'¥0'の位置は変化しません。

戻り値

並べ替えられた文字列を指すポインタを返します。

参照関数

strcpy, strset

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     str[] = "This is a string.";


main()
{
        printf("normal ->%s¥n", str);
        printf("reverse->%s¥n", strrev(str));
}
```

strset

レベル2

書　　式

```
#include <string.h>

char  *strset(string, c);
char  *string;  /*  文字の置き換えが行われる文字列へのポインタ  */
char c;  /*  置き換え文字  */
```

機　　能

指定した文字列stringのすべての文字を、文字cに置き換えます。
ただし末尾のヌル文字'¥0'は変化しません。

戻 り 値

変換された文字列を指すポインタを返します。

参照関数

strnset

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     str[] = "This is a string.";


main()
{
        printf("normal ->%s¥n", str);
        printf("convert->%s¥n", strset(str, '@'));
}
```

strsfn

レベル 2

書　式

```
#include <string.h>

void strsfn(fname, drv, path, node, ext) ;
char  *fname, *drv, *path, *node, *ext ;
```

機　能

ファイル名 fname を分解して、それぞれの引数に格納します。
引数との対応は次の通りです。

drv　：ドライブ名
path ：パス名
node ：主ファイル名
ext　：拡張子

戻り値

戻り値はありません。

参照関数

strmfe

strspn

レベル2

書式

include <string. h>

size_t strspn(string1, string2) ;
char *string1 ; /* 検索が行われる文字列へのポインタ */
char *string2 ; /* 文字セットへのポインタ */

機能

string1の先頭から、string2に含まれる文字が何文字続いているかを調べます。

戻り値

何文字続いていたかをバイト数で返します。
最初からstring2に含まれる文字がなければ0を返します。
すべてがstring2に含まれている文字ならばstring1の全長（ヌル文字は含めない）を返します。

参照関数

strcspn

プログラム例

```
#include        <stdio.h>
#include        <string.h>

char    str[] = "sharp Computer X6800.";
char    set[] = "_ abcdefghijklmnopqrstuvwxyz";


main()
{
        printf("string       ->%s¥n", str);
        printf("sufix length ->%d¥n", strspn(str, set));
}
```

strsrt

レベル 2

書　式

#include <string. h>

void strsrt(s, n) ;
char ＊s[ ] ; /＊ 文字列へのポインタ配列 ＊/
int n ; /＊ 配列の要素数 ＊/

機　能

sが指す文字列の配列をASCIIコード順にソートします。
nには配列の要素数を指定します。

配列のサイズが大きいときにはtqsort関数を使用したほうが効率的です。
また、getfnl、strbpl関数と組み合わせて使うと便利です。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <string.h>


main(int argc, char *argv[])
{
        int     i;

        for (i = 0; i < argc; i++)
                printf("%s¥n", argv[i]);

        printf("sorting...¥n");
        strsrt(argv, argc);

        for (i = 0; i < argc; i++)
                printf("%s¥n", argv[i]);

}
```

strstr　レベル 2

書　式　#include <string. h>

char　＊strstr(string1, string2)；
char　＊string1；　/＊　検索対象文字列へのポインタ　＊/
char　＊string2；　/＊　検索文字列へのポインタ　＊/

機　能　string1 で指定された検索対象文字列の中から、string2 で指定された検索文字列に最初に一致するところのポインタを返します。

戻り値　検索文字列が見つかった場合はそこのポインタを返します。
見つからなかった場合は NULL を返します。

プログラム例

```
#include        <stdio.h>
#include        <string.h>


main()
{
        char    *str;
        char    *pat;
        char    *p;

        str = "This is an apple, and this is a banana.";
        pat = "an";

        printf("%40s¥n", str);
        while ((p = strstr(str, pat)) != NULL) {
                printf("%40s¥n", p);
                str = p + 1;
        }
}
```

strtod

レベル 2

書　式

```
#include <stdlib.h>

double strtod(str, endptr) ;
char  *str ;  /*  変換する文字列  */
char  **endptr ;  /*  変換終了位置を返すためのポインタ  */
```

機　能

strtod は文字列 str を倍精度(double 型)の浮動小数点値に変換します。
変換できる文字列の形式を以下に示します。

[空白(またはタブ文字)][符号][数字][. 数字][E(または e)[符号] 数字]
[空白(またはタブ文字)]は無視されます。
[符号] は"+"または"−"です。
"+"のときは省略できます。
[数字] は 1 桁以上の 10 進数です。
これがないときは、[. 数字] に少なくとも 1 桁の数字が必要です。
[. 数字] は小数点以下の数字です。
また、−123E−12 のように指数表示することもできます。

strtod は、double の数値として解釈できない文字が現れると、変換をその時点で打ち切ります。
endptr が NULL でなければ、endptr には残りの文字の位置を設定します。
すべて文字が変換できたならば文字列終端のヌル文字'¥0'の位置を、1 文字も変換出来なかったならば、文字列の先頭の位置を設定します。

戻 り 値

変換した double の値を返します。
オーバーフローが発生した場合は 0 を返します。

参照関数

strtol, strtoul, atof

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define STR_A   "1.49597870e11"
#define STR_G   "6.672e-11"


main()
{
        double  a;
        double  g;

        a = strtod(STR_A, NULL);
        printf("%s -> %g¥n", STR_A, a);
        g = strtod(STR_G, NULL);
        printf("%s -> %g¥n", STR_G, g);
}
```

strtok

レベル 2

書　　式　#include <string. h>

```
char    *strtok(string1, string2) ;
char    *string1 ;  /*  トークンを含む文字列へのポインタ  */
char    *string2 ;  /*  区切り文字を含む文字列へのポインタ  */
```

機　　能　string1から、文字列string2の各要素の文字を区切り文字として区切られる文字列をトークンとして切り出します。
同一文字列から2回以上トークンを切り出す場合、2回目以降のstrtokのstring1にはNULLを指定します。
このようにすると、次のトークンを検索することができます。
また区切り文字はstrtok関数を呼び出すたびに変化するので、string2の値は何でもかまいません。

戻 り 値　切り出したトークンを指すポインタを返します。
トークンはすべてヌル文字'¥0'で終了します。
トークンを切り出せなかった場合はNULLを返します。

参照関数　strcspn, strspn

プログラム例

```
#include        <stdio.h>
#include        <string.h>

static char     *str = "data1,data2,data3|abc;def:key word";
static char     *dlm = "¥n¥t ,;:|";


main()
{
        char    *tp;

        tp = strtok(str, dlm);
        printf("token = %s¥n", tp);
        while ((tp = strtok(NULL, dlm)) != NULL)
                printf("token = %s¥n", tp);
}
```

strtol

レベル2

書　　式

```
#include <stdlib.h>

long strtol(str, endptr, radix) ;
char   *str ;  /*  変換する文字列   */
char   **endptr ;  /*  変換終了位置を返すためのポインタ   */
int radix ;  /*  指数指定   */
```

機　　能

strtolは文字列strを進数radixにしたがって、符号付き整数に変換します。
radixは、文字列が何進数で表現されているかを指定します。
指定可能な範囲は、0か2～36です。
変換できる文字列の形式は、radixの指定により異なります。
それぞれの場合に分けて、以下に示します。

● radixが0のとき

[空白(またはタブ文字)] [符号] [0で始まらない10進数字]
[空白(またはタブ文字)] [符号] [0で始まる8進数字]
[空白(またはタブ文字)] [符号] [0B(または0b)で始まる2進数字]
[空白(またはタブ文字)] [符号] [0X(または0x)で始まる16進数字]
[空白(またはタブ文字)] は無視されます。
[符号] は"+"または"−"です。
"+"のときは省略されます。
[10進数字] は、0～9までの文字です。
[8進数字] は、0～7までの文字です。
[2進数字] は、0か1の文字です。
[16進数字] は、0～9、A～F(またはa～f)の文字です。

● radixが2のとき

[空白(またはタブ文字)] [符号] [0B(または0b)で始まる2進数字]
[空白(またはタブ文字)] は無視されます。
[符号] は"+"または"−"です。"+"のときは省略されます。
[0B(または0b)] は、無くてもかまいません。
[2進数字] は、0か1の文字です。

● radixが16のとき

[空白(またはタブ文字)] [符号] [0X(または0x)で始まる16進数字]
[空白(またはタブ文字)] は無視されます。
[符号] は"+"または"−"です。"+"のときは省略されます。
[0X(または0x)] は、無くてもかまいません。

[16進数字] は、0~9、A~F(または a~f)の文字です。

● radix が 0、2、16 以外のとき

[空白(またはタブ文字)] [符号] [指定数字]

[空白(またはタブ文字)] は無視されます。

[符号] は"+"または"−"です。"+"のときは省略されます。

[指定進数字] は、0~9、A~Z(または a~z)の文字の最初から radix 個の文字です。

strtol は、指定の進数字の範囲外の文字が現れると、変換をその時点で打ち切ります。

endptr が NULL でなければ、endptr には残りの文字の位置を設定します。

すべての文字が変換できたならば文字列終端のヌル文字'¥0'位置を、一文字も変換出来なかったならば文字列の先頭の位置を設定します。

radixが0のときは、010は8進数の10、0B10は2進数の10として変換します。

戻り値

変換した long の値を返します。

オーバーフローが発生した場合、負の方向でのオーバーフローならば LONG_MIN を、正の方向でのオーバーフローならば LONG_MAX を返し、それとともに errno に ERANGE を設定します。

1 文字も変換できなかった場合は 0 を返します。

参照関数

strtoul, strtod

プログラム例

```
#include        <stdio.h>
#include        <string.h>


main()
{
        char    *rp;
        long    l;

        l = strtol("123", &rp, 10);
        printf("long = %ld, p->%s¥n", l, rp);
        l = strtol("123", &rp, 8);
        printf("long = %ld, p->%s¥n", l, rp);
        l = strtol("123", &rp, 2);
        printf("long = %ld, p->%s¥n", l, rp);
}
```

strtoul

レベル2

書　式

```
#include <stdlib.h>

unsigned long strtoul(str, endptr, radix) ;
char  *str ;  /*  変換する文字列  */
char  **endptr ;  /*  変換終了位置を返すためのポインタ  */
int radix ;  /*  指数指定  */
```

機　能

strtoulは文字列strを進数radixにしたがって符号無し整数に変換します。
radixは、文字列が何進数で表現されているかを指定します。
指定可能な範囲は、0か2～36です。
変換できる文字列の形式は、radixの指定により異なります。
それぞれの場合に分けて、以下に示します。

● radixが0のとき

［空白(またはタブ文字)］［符号］［0で始まらない10進数字］
［空白(またはタブ文字)］［符号］［0で始まる8進数字］
［空白(またはタブ文字)］［符号］［0B(または0b)で始まる2進数字］
［空白(またはタブ文字)］［符号］［0X(または0x)で始まる16進数字］
［空白(またはタブ文字)］は無視されます。
［10進数字］は、0～9までの文字です。
［8進数字］は、0～7までの文字です。
［2進数字］は、0か1の文字です。
［16進数字］は、0～9、A～F(またはa～f)の文字です。

● radixが2のとき

［空白(またはタブ文字)］［符号］［0B(または0b)で始まる2進数字］
［空白(またはタブ文字)］は無視されます。
［0B(または0b)］は、無くてもかまいません。
［2進数字］は、0か1の文字です。

● radixが16のとき

［空白(またはタブ文字)］［符号］［0X(または0x)で始まる16進数字］
［空白(またはタブ文字)］は無視されます。
［0X(または0x)］は、無くてもかまいません。
［16進数字］は、0～9、A～F(またはa～f)の文字です。

● radix が 0、2、16 以外のとき

［空白(またはタブ文字)］［符号］［指定数字］

［空白(またはタブ文字)］は無視されます。

［指定進数字］は、0～9、A～Z(または a～z)の文字の最初から radix 個の文字です。

strtoul は、指定の進数字の範囲外の文字が現れると、その時点で変換を打ち切ります。

endptr が NULL でなければ、endptr には残りの文字の位置を設定します。

すべての文字が変換できたならば文字列終端のヌル文字'¥0'位置を、1 文字も変換出来なかったならば文字列の先頭の位置を設定します。

radixが0のときは、010は8進数の10、0B10は2進数の10として変換します。

戻り値

変換した unsigned long の値を返します。

オーバーフローが発生した場合は ULONG_MAX を返し、それとともに errno に ERANGE を設定します。

1 文字も変換できなかった場合は 0 を返します。

参照関数

strtoul, strtod

プログラム例

```
#include        <stdio.h>
#include        <string.h>


main()
{
        char    *rp;
        unsigned long   ul;

        ul = strtoul("123", &rp, 10);
        printf("long = %lu, p->%s¥n", ul, rp);
        ul = strtoul("123", &rp, 8);
        printf("long = %lu, p->%s¥n", ul, rp);
        ul = strtoul("123", &rp, 2);
        printf("long = %lu, p->%s¥n", ul, rp);
}
```

strupr

レベル2

書式

```
#include <string.h>

char  *strupr(string);
char  *string;  /* 変換が行われる文字列へのポインタ /*
```

機能

文字列 string のすべての英小文字を、英大文字に置き換えます。
他の文字は変化しません。

戻り値

string を指すポインタを返します。

参照関数

strlwr

プログラム例

```
#include        <stdio.h>
#include        <string.h>


main()
{
        char    *p;

        p = "this is lower";
        printf("original  ->%s¥n", p);
        printf("converted ->%s¥n", strupr(p));

        p = "THIS IS UPPER";
        printf("original  ->%s¥n", p);
        printf("converted ->%s¥n", strupr(p));

        p = "This one is $1200.";
        printf("original  ->%s¥n", p);
        printf("converted ->%s¥n", strupr(p));
}
```

swab

レベル2

書式

```
#include <stdlib.h>

void swab(source, destination, n) ;
char  *source; /*  スワップを行うデータへのポインタ  */
char  *destination; /*  スワップ後のデータ格納領域へのポインタ  */
int n; /*  スワップを行うデータのバイト数  */
```

機能

source から n バイトコピーし、隣接するバイトの並び、すなわち偶数アドレスのバイトと奇数アドレスのバイトを入れ替え、その結果を destination にストアします。

n は偶数値でなければなりません。

swab はふつう、異なったバイト並びをもったマシン語へ変換するために使用します。

戻り値

戻り値はありません。

参照関数

swaw

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

#define SIZE    256

static char     src[SIZE];
static char     des[SIZE];


main()
{
        void    setsub(char *);
        void    putsub(char *);

        setsub(src);
        putsub(src);
        swab(src, des, SIZE);
        putsub(des);
}


static void     setsub(char *p)
{
        int     i;

        for (i = 0; i < SIZE; i++)
                *p++ = i;

}


static void     putsub(char *p)
{
        int     i;

        for (i = 0; i < SIZE; i++) {
                printf("%02x ", (*p++ & 0xff));
                if (i % 16 == 15)
                        putchar('¥n');
        }
        putchar('¥n');
}
```

swaw

レベル2

書　式

```
#include <stdlib. h>

viod swaw(src, dst, num) ;
int  *src ; /*  スワップを行うデータへのポインタ  */
int  *dst ; /*  スワップ後のデータ格納領域へのポインタ  */
int num ; /*  スワップを行うデータのバイト数  */
```

機　能

src から num で指定したバイト分コピーして、int 型と同じサイズ（4 バイト）を 1 組とするデータのバイトの並びを逆にします。
結果は、dst に格納されます。
num は 4 の倍数でなければなりません。

戻り値

戻り値はありません。

参照関数

swab

swmem

レベル2

書　　式　#include <string.h>

void swmem(p, q, size);
void　*p;　/*　領域1へのポインタ　*/
void　*q;　/*　領域2へのポインタ　*/
size_t size;　/*　入れ替えるバイト数　*/

機　　能　pが指す領域のsizeバイトと、qが指す領域のsizeバイトの内容を入れ替えます。

戻り値　戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <string.h>


main()
{
        static char     p[] = "Sharp";
        static char     q[] = "x68000";

        printf("p = %s¥n", p);
        printf("q = %s¥n", q);

        swmem(p, q, 4);

        printf("p = %s¥n", p);
        printf("q = %s¥n", q);
}
```

symbol

レベル3

書式

```
#include <graph.h>

void symbol(x, y, st, h, v, mo, p, an) ;
int x ; /* 始点X座標 */
int y ; /* 始点Y座標 */
char *st ; /* 文字列を指すポインタ */
char h ; /* 横倍率 */
char v ; /* 縦倍率 */
int mo ; /* フォントの種類 */
int p ; /* パレットコード */
int an ; /* 回転 */
```

機能

グラフィック画面上に文字列を表示します。
クリッピング領域は、window関数で指定した範囲になります。
xには始点X座標を指定します（−32768～32767）。
yには始点Y座標を指定します（−32768～32767）。
pにはパレットコードを指定します。
moにはフォントの種類を指定します。

0：6×12
1：8×16
2：12×24

anには回転数を指定します（0～3）。
1単位は90°です。
stには表示文字列を指すポインタを指定します。
hには横倍率を指定します。
vには縦倍率を指定します。

戻り値

戻り値はありません。

プログラム例

```
#include        <stdio.h>
#include        <conio.h>
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     mo, pc, an;
        int     hm, vm;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);

        cputs("horizontal multiple (0-255) hm¥r¥n  hm = ?");
        scanf("%d", &hm);

        cputs("vertical multiple (0-255) vm¥r¥n  vm = ?");
        scanf("%d", &vm);

        cputs("character pattern size mo¥r¥n");
        cputs("0:  6 x 12¥r¥n");
        cputs("1:  8 x 16¥r¥n");
        cputs("2: 12 x 24¥r¥n");
        cputs("mo = ?");
        scanf("%d", &mo);

        cputs("color (0-15) pc¥r¥n  pc = ?");
        scanf("%d", &pc);

        cputs("angle an¥r¥n");
        cputs("0:   0 deg¥r¥n");
        cputs("1:  90 deg¥r¥n");
        cputs("2: 180 deg¥r¥n");
        cputs("3: 270 deg¥r¥n");
        cputs("an = ?");
        scanf("%d", &an);

        symbol(384, 255, "x68000", hm, vm, mo, pc, an);
        while(kbhit() == 0)
                ;

        b_exit(0);
}
```

system

レベル２

書　　式

#include <stdlib. h>

int system(string) ;
char　*string ;　/*　Human68k コマンドの文字列を指すポインタ　*/

機　　能

文字列 string を Human68k のコマンドとしてコマンドプロセッサ（COMMAND. X）にわたします。
system は、COMMAND. X を探すために path の環境変数を参照します。

戻 り 値

正の数（0 も）で正常終了、−1 でエラー。
正の数の場合、返り値は起動した子プロセス（COMMAND. X）の終了コード。

参照関数

execl, execle, execlp, execv, execve, execvp, exit, _ exit, spawnl, spawnle, spawnlp, spawnv, spawnve, spawnvp

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>

main(int argc, char *argv[])
{
        char    *str;

        str = (argc == 1) ? "date" : argv[1];

        if (system(str) < 0) {
                printf("error¥n");
                exit(1);
        }
        exit(0);
}
```

tan, tanh

レベル2

書　式

```
#include <math.h>

double tan(x) ;
double tanh(x) ;

double x ;
```

機　能

tanは、xのタンジェントを、tanhはxのハイパボリック（双曲線）タンジェントを計算します。

戻り値

tanはxのタンジェント値を返します。
tanhはハイパボリックタンジェント値を返します。

参照関数

acos, asin, atan, atan2, cos, cosh, sin, sinh

プログラム例

```
#include        <stdio.h>
#include        <math.h>


main()
{
        double  x;
        int     i;

        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("tan(%+4.1f) = %g¥n", x, tan(x));
        }
        for (i = -32; i <= 32; i++) {
                x = (double)i / 10.0;
                printf("tanh(%+4.1f) = %g¥n", x, tanh(x));
        }
}
```

tell

レベル 2

書　　式

#include <io. h>

long tell(handle) ;
int handle ; /* 該当ファイルのファイルハンドル */

機　　能

ファイルハンドルhandleのファイルポインタの現在位置を調べます。

戻 り 値

現在の位置を先頭からのバイト数で返します。

ファイルハンドルの引数が無効の場合は−1を返し、errnoにEBADFがセットされます。

シーク不可能なデバイス（RS-232Cやプリンタなど）を指定した場合、戻り値は不定となります。

参照関数

ftell, lseek

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <io.h>
#include        <stat.h>

#define LEN             100L


main(int argc, char *argv[])
{
        char    buf[LEN];
        char    *file;
        int     fn;

        file = (argc > 1) ? argv[1] : "file";
        if ((fn = open(file, O_RDONLY|O_BINARY)) == -1) {
                perror(file);
                exit(1);
        }
        if (lseek(fn, -LEN, SEEK_END) == -1L) {
                perror("seek error");
                exit(1);
        }
        printf("pos = %ld¥n", tell(fn));
        close(fn);
        exit(0);
}
```

tempfile

レベル2

書　式

#include <stdio.h>

FILE　*tempfile(dir, prefix)；
char　*dir；　/*　作成するファイルのパス名の部分　*/
char　*prefix；　/*　作成するファイルの基本部分　*/

機　能

tempfileは、テンポラリファイルをバイナリモードで更新用("w+b")にオープンします。
作成されるテンポラリファイルは、クローズしたり、プログラムがexit関数によって正常終了した時点で、自動的に削除されます。
tempfileは、オープンされたファイルが自動的に削除されることを除けば、fopen(tempnam(dir, prefix), "w+b")と同じ動作をします。
dirやprefixについては、tempnam関数を参照して下さい。
tempfileは、ファイル名を内部の静的な領域に作成し、そのファイル名を用いてファイルをオープンします。
この静的な領域はtempnamやtmpnam関数などでも使用するため、tempfileを呼び出すと、それらの関数が作成したファイル名を消してしまうので注意して下さい。

戻り値

オープンに成功すると、オープンしたストリームのポインタを返します。
オープンに失敗すると、ヌルポインタを返します。

参照関数

tmpfile, tempnam, tmpnam

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        FILE    *fp;

        if ((fp = tempfile("b:", "sample.tmp")) == NULL)
                fprintf(stderr, "can't open temp file¥n");
        exit(0);
}
```

tempnam
レベル2

書式 #include <stdio.h>

```
char   *tempnam(dir, prefix) ;
char   *dir ;  /*  作成するファイル名のパス名の部分  */
char   *prefix ;  /*  作成するファイル名の基本部分  */
```

機能 tempnamは、既存のファイルとは異なるファイル名を作成します。
作成されたファイル名は、一時的に使用するテンポラリファイルの名前として使用することができます。
tempnamは、最大TMP_MAX回まで異なるファイル名を作成します。
dirには、パス名を指定します。
作成されるファイル名のパス名はdirになります。
prefixは、ファイル名の基本部分を指定します。
作成されるファイル名の主ファイル名は、prefixに付加した文字列になります。
prefixに拡張子が含まれていると、作成されるファイル名の拡張子はそれになります。
prefixに拡張子が含まれていないと、拡張子は"$$$"になります。
dirが、カレントドライブ、カレントディレクトリを参照する(ドライブ名が含まれていない、またはパス名が含まれていない、相対パスを使用している場合など)ときは、次のことに注意してください。
作成した名前でファイルをオープンする前に、カレントドライブカレントディレクトリを変更すると、作成したファイル名は意味をなさなくなってしまいます。
tempnamは、ファイル名を内部の静的な領域に作成し、そのポインタを返します。
そのため、再度tempnamやtmpnamなどを呼び出すと、上書きして古いファイル名を消してしまいます。
この関数によって作成されたファイル名で普通に(fopenやopen関数などで)ファイルを作成した場合、作成されたファイルは、自動的に消されることはありません。
プログラマの責任によって削除してください。

戻り値 作成に成功すると、作成した文字列のポインタを返します。
作成に失敗すると、NULLを返します。

参照関数 tmpfile, tempfile, tmpnam

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        char    *name;
        FILE    *fp;

        if ((name = tempnam("B:", "sample.tmp")) == NULL)
                fprintf(stderr, "can't create temp file¥n");
        if ((fp = fopen(name, "w")) == NULL) {
                fprintf(stderr, "can't open temp file¥n");
                exit(1);
        }
        else {

                fprintf(fp, "hello¥n");
                system("dir ¥¥");
                system("dir");
                (void)fclose(fp);
                exit(0);
        }
}
```

time

レベル2

書式	#include <time.h> time_t time(tp); time_t *tp;
機能	システムクロックが、グリニッジ標準時（GMT）1970年1月1日0時0分0秒から経過した秒数を求めます。 tpがNULLでなければ秒数は、tpにも格納されます。
戻り値	経過秒数を返します。
参照関数	asctime, ftime, gmtime, localtime, utime, ctime

tmasc

レベル 2

書　式

#include <io. h>

char　*tmasc(time)；
int　time；　/*　時刻情報　*/

機　能

timeで指定された時刻情報を文字列に変換し、その文字列へのポインタを返します。

timeの指定は以下の通りです。

ビット 15～11　：時(0～23)
ビット 10～5　：分(0～59)
ビット 4～0　：秒(0～59/2)

timeの形式はtmcnv関数の戻り値と同じです。

timeより変換・作成される文字列は、内部の静的なバッファに格納されるため、この関数を呼び出すと以前の内容は失われることに注意してください。

戻り値

正常終了ならば、内部の静的バッファに作成・格納された文字列へのポインタを返します。

返される文字列は、時をHH、分をMM、秒をSSとすると、"HH：MM：SS"という形式となります。

エラーならばNULLを返します。

参照関数

tmcnv

プログラム例

```
#include        <stdio.h>
#include        <io.h>
#include        <fcntl.h>
#include        <stat.h>


main()
{
        char    *p;
        int     fn;
        int     datetime;

        fn = open("¥¥CONFIG.SYS", O_RDWR);
        datetime = getft(fn);

        if ((p = dtasc(datetime >> 16)) != NULL)
                printf("日付: %s¥n", p);
        if ((p = tmasc(datetime & 0xffff)) != NULL)
                printf("時刻: %s¥n", p);
        close(fn);
}
```

tmcnv

レベル２

書　式

#include <io.h>

int tmcnv(timestr) ;
char *timestr ; /* 変換したい時刻文字列 */

機　能

timestrで指定された文字列を時刻情報に変換して、その値を返します。
timestrの指す文字列は、時をHH、分をMM、秒をSSとすると、"HH：MM：SS" という形式で指定します。
timestrは、tmasc関数の戻り値と同じです。

戻り値

正常終了ならば、以下に示すフォーマットで時刻情報を返します。

ビット 15～11 ：時(0～23)
ビット 10～5 ：分(0～59)
ビット 4～0 ：秒(0～59/2)

エラーならば負の数を返します。

参照関数

tmasc

プログラム例

```
#include        <stdio.h>
#include        <io.h>
#include        <fcntl.h>
#include        <stat.h>

main()
{
        int     fn;
        int     datetime;
        int     date;
        int     time;

        if ((date = dtcnv("90-07-14")) < 0)
                printf("エラー¥n");
        if ((time = tmcnv("12:30:00")) < 0)
                printf("エラー¥n");

        datetime = ((date << 16) | time);

        fn = open("¥¥CONFIG.SYS", O_RDWR);
        chgft(fn, datetime);
        close(fn);
}
```

tmpfile

レベル2

書　式

#include <stdio. h>

FILE　*tmpfile( )；

機　能

tmpfileは、テンポラリファイルをバイナリモードで更新用("w+b")にオープンします。

作成されるファイルは、クローズしたり、プログラムがexit関数によって正常終了した時点で自動的に消去されます。

tmpfileは、オープンされたファイルが自動的に消去されることを除けば、fopen(tmpnam(NULL), "w+b")と同じ動作をします。

tmpfileは、ファイル名を内部の静的な領域に作成し、そのファイル名を用いてファイルをオープンします。

この静的な領域はtempnamやtmpnam関数などでも使用するため、tmpfileを呼び出すと、それらの関数が作成したファイル名を消してしまうので注意してください。

戻り値

オープンに成功すると、オープンしたストリームのポインタを返します。

オープンに失敗すると、NULLを返します。

参照関数

tmrpfile, tempnam, tmpnam

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        FILE    *fp1;
        FILE    *fp2;

        if ((fp1 = tmpfile()) == NULL)
                fprintf(stderr, "can't open temp file¥n");
        if ((fp2 = tmpfile()) == NULL)
                fprintf(stderr, "can't open temp file¥n");

        fprintf(fp1, "hello");
        fprintf(fp2, "hello");
        system("dir ¥¥");
        system("dir");

        (void)fclose(fp2);
        (void)fclose(fp1);
        exit(0);
}
```

tmpnam

レベル２

書式

```
#include <stdio.h>

char  *tmpnam(str);
char  *str;  /*  作成したファイル名を格納する領域へのポインタ  */
```

機能

tmpnamは、既存ファイルとは異なるファイル名を作成します。
作成されたファイル名は、一時的に使用するテンポラリファイルの名前として使用することができます。
tmpnamは、最大TMP_MAX回まで異なるファイル名を作成します。
strは、作成したファイル名を格納する領域へのポインタか、NULLを指定します。
領域を指定した場合、その領域は最低でもL_tmpnam個の文字が格納できなければなりません。
NULLを指定した場合、ファイル名は内部の静的な領域に作成されます。
そのため、再度tempnamやtmpnamなどを呼び出すと、上書きして古いファイル名を消してしまいますので注意してください。
この関数が作成するファイル名は、カレントドライブ、カレントディレクトリを変更すると、意味をなさなくなってしまうので注意してください。
この関数によって作成されたファイル名で普通に（fopenやopen関数などで）ファイルを作成した場合、作成されたファイルは、自動的に消されることはありません。
プログラマの責任によって削除してください。

戻り値

作成に成功したとき、strがNULLの場合、内部の静的な領域のポインタを返し、strがNULLではない場合、strを返します。
作成に失敗すると、NULLを返します。

参照関数

tempnam, tempfile, tmpfile

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        char    buf[L_tmpnam];
        char    *name;
        FILE    *fp;

        if ((name = tmpnam(buf)) == NULL)
                fprintf(stderr, "can't create temp file¥n");
        if ((fp = fopen(name, "w")) == NULL)
                fprintf(stderr, "can't open temp file¥n");

        fprintf(fp, "hello¥n");
        system("dir ¥¥");
        system("dir");

        (void)fclose(fp);
        exit(0);
}
```

toascii, tolower, toupper _tolower, _toupper

レベル 2

書式

```
#include <ctype.h>

int toascii(c);
int tolower(c);
int _tolower(c);
int toupper(c);
int _toupper(c);

int c; /* 文字 */
```

機能

文字 c を次のように変換します。

【toascii】
文字 c の下位 7 ビット以外を 0 にして、ASCII 文字に変換します。
c が ASCII 文字を表している場合は変換しません。

【tolower】
文字 c が大文字の場合は小文字に変換します。
大文字以外の場合は変換しません。

【_tolower】
文字 c が大文字の場合は小文字に変換します。
大文字以外の場合は不定です。

【toupper】
文字 c が小文字の場合は大文字に変換します。
小文字以外の場合は変換しません。

【_toupper】
文字 c が小文字の場合は大文字に変換します。
小文字以外の場合は不定です。

戻り値

変換後の文字を返します。
エラー時の戻り値はありません。

参照関数　isalnum, isalpha, isascii, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit

プログラム例

```
#include        <stdio.h>
#include        <ctype.h>


main()
{
        int     c;
        int     c1, c2, c3, c4;

        printf("     c  low  upp _low _upp¥n");
        for (c = 0x20; c < 0x7f; c++) {
                c1 = tolower(c);
                c2 = toupper(c);
                c3 = isupper(c) ? tolower(c) : c;
                c4 = islower(c) ? toupper(c) : c;
                printf("%02x:  %c    %c    %c    %c    %c¥n",
                  c, c, c1, c2, c3, c4);
        }
}
```

注：以上はマクロ定義と関数版があります。マクロ定義にはctype. hを用い、関数版にはfctype. hを用いてください。

tzset　　レベル2

書　式

```
#include <time.h>

void tzset( );

int daylight;
time_t timezone;
char  *tzname [2];
```

機　能

環境変数 TZ に設定されている値を利用して、次の変数に値を代入します。

timezone	：グリニッジ標準時と地方時の時差秒数
tzname [0]	：3文字のタイムゾーン文字列
tzname [1]	：3文字の夏時間調節ゾーン文字列
daylight	：tzname [1] が空文字列（NULL）ならば0 tzname [1] が空文字列（NULL）でなければ夏時間調節秒数

これらの変数は、地方時を扱う関数においてグリニッジ標準時（GMT）を地方時に変換するときに使います。
なお、プログラム起動時に自動的に tzset が呼び出されます。
環境変数 TZ には、以下のような値を設定しておきます。

JST−9（日本の場合）

たとえば ABC5DEF1 ならば、タイムゾーン文字列は“ABC”、夏時間調節ゾーン文字列は“DEF”、時差は5時間、夏時間調節は1時間になります。
また、夏時間調節文字列の後の数字を省略するか、0にすると1時間になります。
また、夏時間調節文字列がない場合は、夏時間はなしになります。
環境変数 TZ がない場合は、JST−9（日本時間）になります。
環境変数 TZ の設定は、Human68K のコマンドラインで以下のようにして行うことができます。

A>SET　TZ=GMT0 ↵

なお、時差は、グリニッジ標準時−地方時で求めます。
たとえば、グリニッジ標準時が3時のとき地方時が13時であれば、3−13=−10で、−10時間になります。

戻り値	戻り値はありません。
参照関数	asctime, ftime, localtime

uitoa

レベル2

書　式　#include <stdlib. h>

char　*uitoa(val, str, rdx) ;

unsigned int val ;　/*　変換する数値　*/
char　*str ;　/*　文字列を格納する領域　*/
int rdx ;　/*　基数　*/

機　能　数値 val を文字列 str に変換します。
rdx は val が何進数なのかを表し（基数という）、必ず 2 から 36 の範囲で指定します。
str は変換後の文字列が十分納まる大きさでなければなりません。

戻り値　str を指すポインタを返します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        char    buf[34];
        unsigned int    ui;

        for (ui = 0; ui < 60000U; ui += 1000) {
                printf("%4u : ", ui);
                printf("%7s ", uitoa(ui, buf, 2));
                printf("%3s ", uitoa(ui, buf, 8));
                printf("%2s ", uitoa(ui, buf, 10));
                printf("%2s ", uitoa(ui, buf, 16));
                printf("¥n");
        }
}
```

ultoa

レベル2

書　式

#include <stdlib. h>

char　＊ultoa(value, string, radix)；
unsigned long value；　/＊　変換すべき整数値　＊/
char　＊string；　/＊　文字列の格納領域を指すポインタ　＊/
int radix；　/＊　基数　＊/

機　能

数値valueをヌル文字'¥0'で終了する文字列に変換し、stringに格納します。
オーバーフローのチェックは行われません。
引数radixは、valueの基数で、2≦radix≦36の整数です。
文字列stringを格納するためには、最大33バイトの領域が必要です。

戻り値

stringを指すポインタを返します。

参照関数

itoa, ltoa

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        char    buf[34];
        unsigned long   ul;

        for (ul = 0; ul < 1000000UL; ul += 10000UL) {
                printf("%10lu : ", ul);
                printf("%20s ", ultoa(ul, buf, 2));
                printf("%8s ", ultoa(ul, buf, 8));
                printf("%7s ", ultoa(ul, buf, 10));
                printf("%6s ", ultoa(ul, buf, 16));
                printf("¥n");
}
}
```

ungetc

レベル 2

書　式

```
#include <stdio.h>

int ungetc(c, stream) ;
int c; /* 返される文字 */
FILE *stream; /* FILE構造体へのポインタ */
```

機　能

入力ストリームstreamから、直前に読み込んだ文字cを入力ストリームに戻し、ファイルポインタを1文字分戻します。

streamはバッファリングされ、読み込みのためにオープンしていなければなりません。

なお、ungetcを使ってEOFを戻そうとしても無視されます。

また、streamから何も読み込んでいない場合や、cが戻すことができない場合は、エラーとなります。

入力ストリームに戻した文字は、fseekやrewind関数を実行すると消されます。

戻り値

入力ストリームに戻した文字を返します。

エラーが発生した場合はEOFを返します。

参照関数

getc, petchar, putc, putchar

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <ctype.h>


main()
{
        void    out_digit(FILE *);
        FILE    *fp;
        int     c;

        while ((c = getchar()) != EOF) {
                if (isdigit(c)) {
                        ungetc(c, stdin);
                        out_digit(stdin);
                }
        }
        exit(0);
}

static void     out_digit(FILE *fp)
{
        int     c;

        while ((c = getchar()) != EOF) {
                if (isdigit(c))
                        putchar(c);
                else {
                        ungetc(c, fp);
                        break;
                }
        }
}
```

ungetch

レベル 2

書式

include <conio. h>

int ungetch(c) ;
int c; /* 戻す文字 */

機能

文字cをコンソールに戻し、次の読み込み文字とします。
文字を読み込まないで2回以上連続してungetchを実行するとエラーになります。

戻り値

戻した文字を返します。
エラーが発生した場合はEOFを返します。

参照関数

cscanf, getch, getche

プログラム例

```
#include        <ctype.h>
#include        <conio.h>

#define MAX             64


main()
{
        char    buf[MAX];
        int     i;
        int     c;

        while ((c = getche()), isspace(c))
                ;
        for (i = 0; i < MAX; i++) {
                if (isspace(c))
                        break;
                buf[i++] = c;
                c = getche();
        }
        ungetch(c);
        buf[i] = '¥0';
}
```

unlink, remove　レベル2

書　式

#include <io. h>

int unlink(pathname) ;
char　*pathname;　/*　削除するファイル　*/

#include <stdio. h>

int remove(pathname) ;
char　*pathname;　/*　削除するファイル　*/

機　能

ファイル pathname を削除します。
unlink と remove は定義されているヘッダファイルが異なるだけで、機能、実行結果は同じです。

戻り値

正の数（0も）で正常終了、−1でエラー。
通常は0が返りますが、特殊デバイスドライバが、対象の時0以外の正の数が返ります。

参照関数

close

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <io.h>

main(int argc, char *argv[])
{
        if (argc == 1)
                puts("Usage: unlink filename");
        else {
                if (unlink(argv[1]) == -1) {
                        printf("%sを削除できません。¥n", argv[1]);
                        exit(1);
                }
        }
        exit(0);
}
```

using

レベル3

書　　式　#include <basic0. h>

```
char  *using(s, fmt, ・・・);
char  *s;  /*  変換した文字列を指すポインタ  */
char  *fmt;  /*  書式制御文字列へのポインタ  */
```

機　　能　*fmt の内容により、スタック上のデータが double か string かを判断します。
数値はすべて double です。
なお、書式制御文字は次の通りです。

●数

制御文字	機　　能
#	桁
.	小数点
+｜－	符号
**	*で埋める
¥¥	¥記号をつける
**¥	¥記号をつけ*埋める
^^^^	小数点
,	3桁のカンマ区切り

●文字

制御文字	機　　能
!	先頭の1文字
&	"&"で囲んだだけの文字数分（"&"も含む）文字列を出力する
@	エスケープ文字 これに続く1文字を文字として印刷する

戻 り 値　作成した文字列を指すポインタを返します。

utime

レベル 2

書　　式

#include <utime. h>

```
int utime(pathname, times) ;
char   *pathname ;  /*  該当ファイルのパス名   */
struct utimbuf   *times ;  /*  utimbuf 構造体を指すポインタ   */
```

機　　能

ファイル pathname の更新日時を times にセットします。
ファイルは書き込み可能でなければなりません。
times は utime. h で定義されている utimbuf 型の構造体を指していなければなりません。utimbuf 型構造体は以下の通りです。

```
struct utimbuf {
        time_t actime ;
        time_t modtime ;
} ;
```

変更日時はこの構造体の modtime 領域によってセットされます。
times が NULL の場合は、変更日時は現在の時間にセットされます。

戻 り 値

更新日時を変更できたときには 0 を返します。
エラーが発生した場合は −1 を返し、errno に次の値がセットされます。

errno の値	内　　容
EACCES	pathname がディレクトリを指定、または読み込み専用の属性が付いたファイルを指定している
EMFILE	オープンしているファイルの数が多すぎる
ENOENT	ファイルまたはパス名が見つからない

参照関数

asctime, ctime, fstat, ftime, gmtime, stat, time

プログラム例

```
#include        <stdio.h>
#include        <utime.h>


main()
{
        if (utime("¥¥sales¥¥west", NULL) != 0)
                perror("変更できません。");
}
```

uwtoa, wtoa

レベル2

書式

#include <stdlib. h>

char　*wtoa(wval, str, rdx)；
char　*uwtoa(uval, str, rdx)；

short wval；/*　変換する数値　*/
unsigned short uval；/*　変換する数値　*/
char　*str；/*　文字列格納配列へのポインタ　*/
int rdx；/*　基数　*/

機能

数値 wval, uval を文字列 str に変換します。
rdx は val が何進数なのかを表し（基数という）、必ず 2 から 36 の範囲で指定します。
str は変換後の文字列が十分に納まる大きさでなければなりません。
wtoa で、wval が負の数、rdx が 10 のときは、str の 1 文字目はマイナス（−）になります。

戻り値

str を指すポインタを返します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>


main()
{
        char    buf[34];
        short   s;

        for (s = -100; s < 100; s += 10) {
                printf("%7ld: ", s);
                printf("%032s ", wtoa(s, buf, 2));
                printf("%11s ", wtoa(s, buf, 8));
                printf("%7s ", wtoa(s, buf, 10));
                printf("%8s ", wtoa(s, buf, 16));
                printf("¥n");
        }
}
```

va_start, va_arg, va_end

レベル2

書　式

```
#include <stdarg.h>

viod va_start(param, lastarg) ;
type va_arg(param, type) ;
viod va_end(param) ;
va_list param;
```

機　能

va_start, va_arg, va_end は、可変個の引数の参照を実現するために用意されています。
param は、可変個の引数を参照するための変数です。
可変個の引数を参照する関数の先頭で、va_list 型の変数として宣言します。
va_start は、param が可変個の引数の先頭を指すように設定します。
lastarg は、その関数の固定の引数のうち、最後の引数の名前を設定します。
va_arg は、可変個の引数のうち、param が指す引数を type 型の値として取り出します。
そして、param が次の引数を指すように変更します。
va_end は、可変個の引数の参照を終了させ、va_start, va_arg を使用した関数から呼び出し元へ、正常に制御が戻るように手続きをします。
va_start, va_arg, va_end は、stdarg. h でマクロとして宣言されています。

戻り値

va_start, va_end は、戻り値がありません。
va_arg は、可変個の引数のうち、param が指す引数を、type 型の値として返します。

参照関数

vfprintf, vprintf, vsprintf

プログラム例

```
#include        <stdio.h>
#include        <stdarg.h>
#include        <string.h>

main()
{
        char    *addstr(char *, ...);
        char    buf[256];
        char    *ptr;

        ptr = addstr(buf, "abc", "123", "ABC", NULL);
        printf("%s¥n", ptr);
        ptr = addstr(buf, "123", "ABC", "ABC", "XYZ", NULL);
        printf("%s¥n", ptr);
}

static char     *addstr(char *buf, ...)
{
        va_list ap;
        char    *s;

        va_start(ap, buf);
        *buf = '¥0';
        while ((s = va_arg(ap, char *)) != NULL)
                strcat(buf, s);
        va_end(ap);
        return (buf);
}
```

val

レベル3

書　式

```
#include <basic.h>

double val(s) ;
char  *s;  /*  変換を行う文字列へのポインタ  */
```

機　能

文字列を数値に変換します。
数値には、X-BASIC の表現（&H など）を使うこともできます。

戻り値

変換後の数値を返します。

v_cut　レベル 3

書　式

```
#include <image.h>

void v_cut(tv_sw) ;
char tv_sw;  /*  制御スイッチ  */
```

機　能

スーパーインポーズ表示時、テレビ画面の表示制御を行います。
tv_sw に指定できる内容は次の通りです。

0 : テレビ画面の表示
1 : テレビ画面の表示カット

また、次の関数の組み合わせにより、各種の画面を表示することができます。

v_cut (0) : vpage (1) …… スーパーインポーズ
v_cut (0) : vpage (0) …… テレビ画面
v_cut (1) : vpage (1) …… コンピュータ (グラフィック)

戻り値

戻り値はありません。

vfprintf, vprintf, vsprintf, vcprintf　レベル 2

書　式

```
#include <stdio.h>

int vfprintf(stream, format, argptr);
int vprintf(format, argptr);
int vsprintf(buffer, format, argptr);
FILE   *stream; /*  出力するファイルストリーム  */
char   *format; /*  出力書式  */
char   *buffer; /*  出力先文字列バッファ  */
va_list argptr; /*  引数へのポインタ  */

#include <conio.h>

int vcprintf (format, argptr);
char   *format; /*  出力書式  */
va_list argptr; /*  引数へのポインタ  */
```

機　能

vfprintf, vprintf, vsprintf, vcprintf 関数は、それぞれ fprintf, printf, sprintf, cprintf とほぼ同じ機能を持つ関数です。

これらの違いは、出力したい引数を直接指定するのではなく、引数リストへのポインタを指定する点です。

各関数の詳細は、それぞれに対応する関数を参照してください。

動作については、そちらを参照して下さい。

引数リストへのポインタ argptr は、va_start 関数によって設定される可変個の引数を参照するための変数を指定します。

例えば、printf 関数を vprintf を用いて実現すると、以下のようになります。

```
int printf(format)
char   *format;
 {
        va_list argptr;
        int status;
        va_start(argptr, format);
        status = vprintf(format, argptr);
        va_end(argptr);
        return(status);
}
```

戻り値	出力した文字数を返します。 エラーの時は負の数を返します。
参照関数	fprintf, printf, sprintf, va_start, va_end, cprintf

プログラム例

```
#include        <stdio.h>
#include        <stdarg.h>


main()
{
        int     eprintf(char *);

        printf("これは標準出力です¥n");
        eprintf("これは標準エラー出力です¥n");
}


static int      eprintf(char *format)
{
        va_list argptr;
        int     status;

        va_start(argptr, format);
        status = vfprintf(stderr, format, argptr);
        va_end(argptr);
        return (status);
}
```

vpage

レベル3

書　式

```
#include <graph.h>

void vpage(i) ;
char i;  /*  表示ページの指定  */
```

機　能

グラフィック画面の各ページ（0～3）の表示をするかしないかを設定します。
iのビット0～3がページ0～3にそれぞれ対応し、1（表示ON）または0（表示OFF）で表します。
また、iは次のような範囲の値になります。

スクリーンモード	i	ページ数
1024×1024（16色）	0，1	1
512×512（16色）	0～15	4
512×512（256色）	0～3	2
512×512（65536色）	0，1	1

戻り値

戻り値はありません。

wait

レベル 2

書　　式

```
#include <process.h>

int wait( );
```

機　　能　現在、実行しているプログラムが直前に実行した子プロセスの終了コードを受け取ります。

戻 り 値　子プロセスの終了コードを返します。

width

レベル3

書　式

```
#include <basic0.h>

void width(n);
int n;  /*  1行の表示文字数  */
```

機　能

画面の1行の表示文字数を設定します。
nは64か96のどちらかを設定してください。

戻り値

戻り値はありません。

window　　レベル3

書　式　#include <graph. h>

void window (x1, y1, x2, y2) ;
int x1; /* 始点 X 座標 */
int y1; /* 始点 Y 座標 */
int x2; /* 終点 X 座標 */
int y2; /* 終点 Y 座標 */

機　能　グラフィック画面のクリッピング領域の座標を指定します。
x1 には始点 X 座標を指定します。
y1 には始点 Y 座標を指定します。
x2 には終点 X 座標を指定します。
y2 には終点 Y 座標を指定します。
上記の座標の指定可能範囲は、実画面サイズにより 0～511、または 0～1023 となります。
座標を指定するときは x1≦x2、y1≦y2 となってなければいけません。

戻り値　戻り値はありません。

参照関数　screen

プログラム例

```
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);

        wipe();
        window(0, 0, 767, 511);
        line(0, 0, 767, 511, 15, 0xffff);

        for (i = 0; i < 3000; i++)
                ;

        wipe();
        window(0, 0, 200, 200);
        line(0, 0, 767, 511, 15, 0xffff);

        b_exit(0);
}
```

wipe

レベル3

書　式　#include <graph. h>

void wipe( ) ;

機　能　グラフィック画面をクリアします。

戻り値　戻り値はありません。

参照関数　apage

プログラム例

```
#include        <basic0.h>
#include        <graph.h>


main()
{
        int     i, j;

        b_init();
        screen(2, 0, 1, 1);
        vpage(1);

        fill(0, 0, 767, 511, 15);
        wipe();
        screen(1, 1, 1, 1);
        vpage(0);

        apage(0);
        fill(50, 50, 200, 200, 3);
        apage(1);
        fill(100, 100, 250, 250, 5);
        apage(2);
        fill(150, 150, 300, 300, 9);
        apage(3);
        fill(200, 200, 350, 350, 15);
        vpage(15);
        for (i = 0; i < 4; i++) {
                for (j = 0; j < 3000; j++)
                        ;
                apage(i);
                wipe();
        }
        b_exit(0);
}
```

write

レベル２

書　式

```
#include <io.h>

size_t write(handle, buffer, count);
int handle;  /*  出力するファイルのファイルハンドル  */
void  *buffer;  /*  出力データの格納領域へのポインタ  */
size_t count;  /*  出力バイト数  */
```

機　能

handleで指定されるファイルへbufferからcountバイトを書き込みます。
書き込み処理は、ファイルポインタの現在の位置から始められます。
ファイルを追加用にオープンした場合は、書き込み処理はエンドオブファイル（EOF）から始められます。
書き込み処理の終了後ファイルポインタは、実際に書かれたバイト数だけ進められます。

参照関数

fwrite, open, read

戻り値

実際に書き込んだバイト数を返します。
戻り値がcountより小さい値になる場合もあります（たとえばcountバイトを書く前にディスクに空き領域がなくなった場合)。
エラーが発生した場合は－1を返し、errnoに次の値がセットされます。

errnoの値	内　　容
EACCES	ファイルが読み込み専用
EBADF	ファイルハンドルが無効
ENOSPC	デバイスに空き領域がない

指定されたファイルをテキストモードでオープンした場合、出力時に改行'¥n'は復帰改行'¥r¥n'に置き換えられます。
この置き換えは戻り値には影響を与えません。

テキストモードでオープンされているファイルへ書き込む場合にCTRL-Zコード'¥0x1A'は、論理的エンドオブファイル（EOF）として扱われます。
またバッファ内にCTRL-Zコード文字があると、デバイスへCTRL-Zコードを書き込んだ時点で終了します。

プログラム例

```
#include        <stdio.h>
#include        <stdlib.h>
#include        <fcntl.h>
#include        <io.h>
#include        <stat.h>

#define SIZE    16


main(int argc, char *argv[])
{
        char    buf[SIZE];
        char    *file;
        int     fn;
        int     n;

        file = (argc > 1) ? argv[1] : "file";

        if ((fn = open(file, O_RDONLY|O_TEXT)) == -1) {
                printf("can't open %s¥n", file);
                exit(1);
        }
        while (!eof(fn)) {
                n = read(fn, buf, SIZE);
                (void)write(fileno(stdout), buf, n);
        }
        close(fn);
        exit(0);
}
```

zentohan

レベル 2

書　式

<jfctype. h> を読み込む指令：

```
#include <jfctype. h>

int zentohan(c) ;
int c ;  /*  変換する文字（全角文字）  */
```

機　能

全角文字ｃを半角文字に変換します。

全角文字は、0x20～0x7E、0xA1～0xDF の半角文字に対応する文字でなければなりません。

全角カタカナも全角ひらがなも、半角カタカナに変換されます。

濁点、半濁点の文字は、8～15 ビット目に'ﾞ'、または'ﾟ'のコード、0～7 ビット目に濁点、半濁点なしのカナコードが入ります。

戻り値

半角文字を返します。

変換できなかった場合にはｃの値をそのまま返します。

参照関数

hantozen

プログラム例

```
#include        <stdio.h>
#include        <jfctype.h>

main()
{
        int     ch;
        int     c, cc;

        while ((c = getchar()) != EOF) {
                if (iskanji(c)) {
                        cc = c;
                        ch = (c << 8);
                        ch += (c = getchar());
                        if (jiskata(ch) || jishira(ch)) {
                                putchar(zentohan(ch));
                                putchar(zentohan(ch) >> 8);
                        }
                        else {
                                putchar(cc);
                                putchar(c);
                        }
                }
                else
                        putchar(c);
        }
}
```

付　　録

ユーザーライブラリの作り方

ソースファイルの作成
コンパイル
ライブラリファイル
ライブラリの外部宣言
リンク

C compiler PRO-68Kでは標準ライブラリ、IOCSコールライブラリ、DOSコールライブラリ等を提供していますが、これとは別にユーザー自身がオリジナルにライブラリを作成することができます。
本編では、ユーザーライブラリの作りかたを各ステップごとに述べるとともに、作成にあたっての注意事項についても述べていきます。

1　ソースファイルの作成

ライブラリもメインプログラムと同様に、エディタでソースファイルを作成します。
エディタの起動方法、および操作手順等については、「Human68k ユーザーズマニュアル」を参照してください。

2 コンパイル

ソースファイルの作成が終わったら、コンパイルを行います。コンパイルは、CC コマンドを起動することにより行います。

このとき注意しなければならないことは、CC コマンドを起動すると自動的にリンクまで行ってしまうので、リンク作業を抑止しなければならないということです。

というのは、ライブラリのリンクはメインプログラムとともに行う必要があるので、ライブラリのコンパイル段階でリンクを行うわけにはいかないからです。

そのようなわけで、ライブラリはオブジェクトコードだけ作成しておけば済みます（XC 標準のライブラリもオブジェクトファイルとして提供されています）。

リンク処理の抑止は、Fc スイッチにより行います。ライブラリのコンパイルを行う場合の、CC コマンドのコマンドラインの具体例を以下に示します。

```
CC  /Fc  jstrlen.c ↵
```

なお、CC コマンド、および CC コマンドのスイッチの詳細については、「C ユーザーズマニュアル」の「第２章　コンパイル」を参照してください。

3 ライブラリファイル

C 標準ライブラリの登録ファイルは clib. l です。
ユーザーが自身でライブラリを作成した場合、clib. l に登録することもできますし、またユーザー自身のファイルに登録することもできます。
また、ライブラリはオブジェクトファイル（拡張子 o）の形式でよいのですが、ファイルの管理上、ライブラリファイル（拡張子 l）に登録しておけば何かと便利です。
ライブラリファイルへの登録方法については、「アセンブラマニュアル」の「第 8 章　ライブラリアン」を参照してください。

4 ライブラリの外部宣言

新規に作成したライブラリをメインプログラムから呼び出す場合、ライブラリの外部宣言、および該当ライブラリで使用する（既存のインクルードファイルでは未定義の）記号定数や構造体などがある場合には、これらの定義が必要となります。
上記の外部宣言および定義を行う方法には、以下の２種類があります。

①新規にインクルードファイル（ユーザーインクルードファイル）を作成し、ここで外部宣言と定義を行う。

②メインプログラム中で外部宣言と定義を行う。

新規に作成したライブラリが汎用性に富んでおり、多数のプログラムで使用される可能性がある場合や、そのライブラリが使用する構造体や記号定数が数多い場合などでは、①の方法が有効でしょうし、そうでない場合には、②で済むこともあります。
これについては、ライブラリの作成者が適宜、選択してください。

5　リンク

ライブラリは、メインプログラムのリンク時にはじめて結合されます。
リンク時には、リンカのコマンドライン上にライブラリファイルの名前を指定します。
以下にコンパイラ、およびリンカのコマンドラインの具体例を示します。

```
cc jstrsample. c jstrlen. o ↵
lk jstrsample.o jstrlen.o clib.l floatfnc.l ↵
```

上記の例では、ライブラリファイル名は、clib.l となっていますが、ユーザーがライブラリファイルを作成した場合には、代わりにそのファイル名を指定してください。

索引 50音順

50音順およびアルファベット順の索引は、すべて「Cライブラリマニュアル VOL.1」に対応しています。

ア

イ

エ

オ

カ

ス

セ

ソ

50音順

ヒ

フ

ヘ

50音順

ロ

ワ

アルファベット順

50音順およびアルファベット順の索引は、すべて「Cライブラリマニュアル VOL.1」に対応しています。

F

G

H

I

J

L

M

関数索引 アルファベット順

※小文字関数は「C ライブラリマニュアル VOL.1」、大文字関数は「C ライブラリマニュアル VOL.2」のページ数を表記してあります。

A

B

アルファベット順

D

E

G

H

I

J

K

N

O

P

アルファベット順

W

Z

機能別関数索引

※小文字関数は「C ライブラリマニュアル VOL.1」、大文字関数は「C ライブラリマニュアル VOL.2」のページ数を表記してあります。

バッファ操作

メモリ割りあて

文字の分類と変換

データ変換

ディレクトリ操作

ファイル操作

ストリーム入出力

低水準入出力

コンソール入出力

数値演算

プロセス制御

サーチとソート

文字列操作

strpbrk	指定文字列を探す	504
strrchr	指定文字の最後の位置を調べる	505
strrev	文字列を逆順にする	506
strset	指定文字列に置き換える	507
strspn	指定文字列以外の最初の位置を調べる	509
strstr	char 型引数 1 から char 型引数 2 を検索する	511
strtok	トークンを探す	514
strupr	すべての英小文字を英大文字に変換する	519
strins	文字列を挿入する	496
strmfe	ファイル名の拡張子を置き換える	499
strmfn	ファイル名を作成する	500
strmfp	ファイル名を作成する	500
stcgfe	ファイル名から拡張子をとり出す	484
strbpl	文字列からポインタ配列を作成する	486
stcgfn	ファイル名からノードをとり出す	484
strsrt	文字列を ASCII コード順にソートする	510

時　間

関数名	機　能	ページ
asctime	時間情報を文字列に変換する	108
ctime	時間を int 型整数値から文字列に変換する	207
clock	プロセスの使用時間を求める	194
difftime	時間の引算を行う	208
ftime	システム時間を調べる	271
gmtime	時間を整数値から構造体に変換する	296
localtime	時刻を地方時に変換する	334
mktime	ローカル時間をカレンダ時間に変換する	375
strftime	時間情報を書式にしたがって文字列に変換する	492
time	システム時間を調べる	532
tzset	環境時間変数から外部時間変数を設定する	540

日本語操作

関数名	機　能	ページ
iskana	カナ文字またはカナ句読点の判定を行う	318
iskpun	カナ句読点の判定を行う	318

BASIC 関連

IOCS コール

DOSコール

SLEEP_PR	スレッドをSLEEP状態にする	358
SEND_PR	スレッドにコマンドやデータを送る	344
CHANGE_PR	バックグラウンドタスクの実効権を破棄する	107

その他

関数名	機　能	ページ
assert	論理エラーをテストする	110
getenv	環境変数の値を求める	286
longjmp	セーブされたスタック環境をリストアする	337
offsetof	構造体メンバの位置を求める	395
perror	エラーメッセージを表示する	401
putenv	環境変数の値を修正または追加する	417
abs	int型整数値の絶対値を求める	96
labs	int型整数の絶対値を求める	96
max	2つの引数のうち大きいものを求める	348
min	2つの引数のうち小さいものを求める	369
wabs	short型整数値の絶対値を求める	96
rand	擬似乱数を発生する	426
setjmp	スタック環境をセーブする	451
srand	擬似乱数を初期化する	480
strerror	エラー番号に対応するエラーメッセージを調べる	491
va_arg	可変個の引数を値として読み込む	551
va_end	可変個の引数の参照を終了する	551
va_start	可変個の引数の参照を開始する	551

シャープ株式会社
本　　　社　〒545 大阪市阿倍野区長池町22番22号
電子機器事業本部　〒329-21 栃木県矢板市早川町174番地
液晶映像システム事業部　第2商品企画部
お問い合わせ先　〒162 東京都新宿区市谷八幡町8番地　電話 (03)3260-1161(大代表)
東京支社内　液晶映像システム事業部　第2商品企画部　ソフトウェア担当